LA PRIMERA REVISTA DE PROGRAMACIÓN EN CASTELLANO

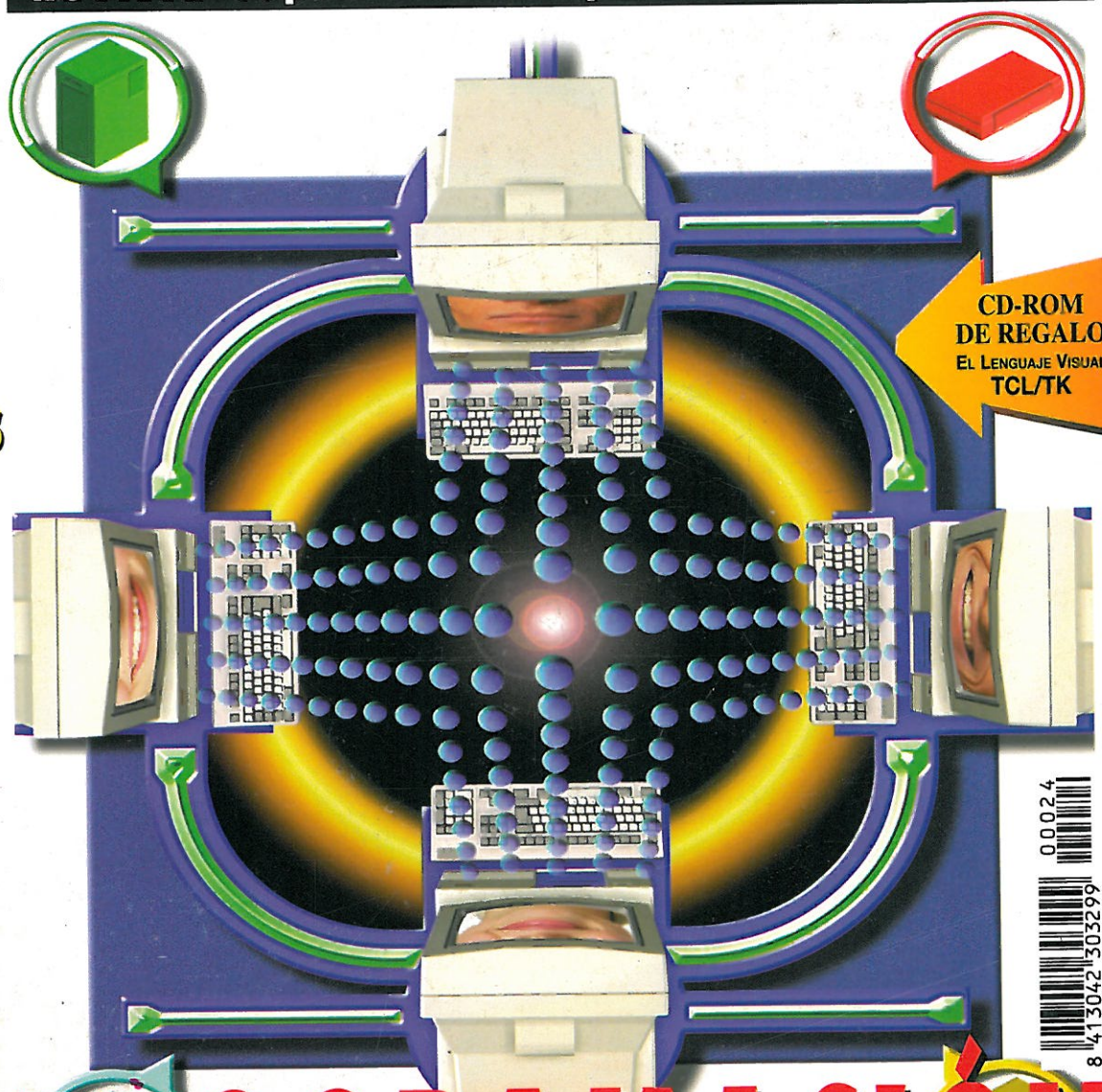
SÓLO PROGRAMADORES

Revista especializada para usuarios de PC

por desarrolladores ESCRITA para desarrolladores

AÑO 3. Nº 24
250 PTAS.

ARGENTINA 9'50 $
CHILE 3000 $
PORTUGAL 1500$



CD-ROM DE REGALO
EL LENGUAJE VISUAL TCL/TK

# PROGRAMACIÓN DEL PUERTO SERIE

ESTUDIO DEL FORMATO WAV

LAS VENTANAS LLEGAN AL WEB

DEPURACIÓN DE CÓDIGO EN VISUAL C++

Y además:

- Redes locales
- La BIOS del PC
- Los Modos X de la VGA

CURSOS PRÁCTICOS DE:

- Creación de un compilador
- Visual Basic 4.0
- Grandes Sistemas
- Programación de una demo

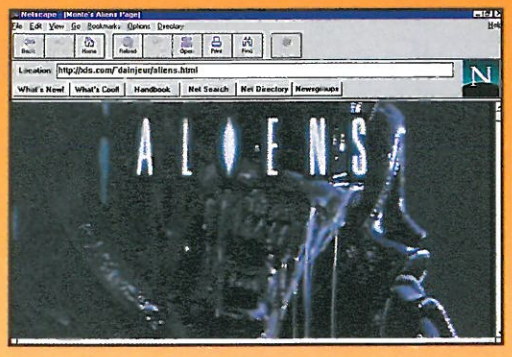
8 413042 303299 00024

TOWER COMMUNICATIONS, S.R.L

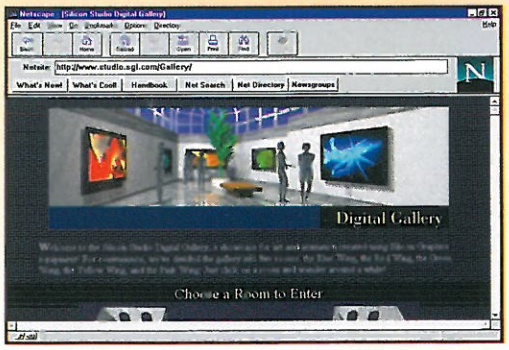
# INTERNET

## INTERNET Directorio

- Ciencia
- Cine
- Comics
- Compras
- Deportes
- Educación
- Filosofía
- Historia
- Humor
- Informática
- Libros
- Música
- Naturaleza
- Ocultismo
- Personajes
- Prensa
- Religión
- Revistas
- Salud
- Sexo
- Tecnología
- Televisión
- Viajes
- Vídeo-Juegos




## Por fin a la venta el 1er Directorio Internet




**Por sólo 1.995 Ptas.**

ABETO EDITORIAL

**Solicita INTERNET DIRECTORIO enviando este cupón o llamando al teléfono (91) 661.42.11* de 9 a 14 y de 15:00 a 18:00 o por Fax: (91) 661.43.86**

Deseo que me envíen: ☐ **INTERNET DIRECTORIO por 1995 ptas. + 250 ptas. de gastos de envío.**

Nombre y apellidos.....Domicilio.....
Población.....C.P .....Provincia.....
Teléfono .....F. de nacimiento.....Profesión.....

**FORMA DE PAGO:**

☐ Talón a ABETO EDITORIAL ☐ Contra-reembolso
☐ Giro Postal nº.....de fecha.....
☐ Tarjeta de crédito VISA nº ☐☐☐☐ ☐☐☐☐ ☐☐☐☐ ☐☐☐☐ CADUCA ☐☐ ☐☐

Firma ,

ABETO EDITORIAL

Rellena este cupón y envíalo a:
ABETO EDITORIAL
C/ Aragoneses, 7
28100 Pol. Ind. ALCOBENDAS (Madrid)

SÓLO PROGRAMADORES
Revista práctica para usuarios de PC

AGOSTO 1996. Número 24
**SÓLO PROGRAMADORES es una publicación de Tower Communications**

**Editor**
Antonio M. Ferrer Abelló

**Director Editorial**
Mario de Luis García

**Subdirector Editorial**
Carlos Doral Pérez

**Director Financiero**
Francisco García Díaz de Liaño

**Director de Producción**
Carlos Peropadre

**Director de Distribución**
Enrique Cabezas García

**Directora Comercial**
Carmina Ferrer

**Director**
Mario de Luis García

**Redactor Jefe**
Carlos Doral Pérez

**Coordinador Técnico**
Eduardo Toribio Pazos

**Edición**
Miguel Cabezuelo

**Colaboradores**
Fernando de la Villa, Fernando J. Echevarrieta, Pedro Antón. Juan M. Martín, Luis Martín, José M. Peco, José C. Remiro, Agustín Guillén, Daniel Navarro, Jorge del Río, Enrique De Alarcón, David Aparicio, M. Jesús Recio, Santiago Romero, Miguel Cubas, Vicente Cubas, Carlos Pérez

**Jefe de Diseño**
Fernando García Santamaría

**Maquetación**
Clara Francés

**Tratamiento de Imagen**
María Arce Giménez

**Servicios Informáticos**
Digital Dreams Multimedia, S.L.

**Secretaría de Redacción**
Rosa Arroyo

**Suscripciones**
Erika de la Riva

**Redacción, Publicidad y Administración**
C/ Aragoneses, 7
28100 Pol. Ind. Alcobendas (MADRID)
Telf.: (91) 661 42 11 / Fax: (91) 661 43 86

**Filmación**
Filma Dos S.L.

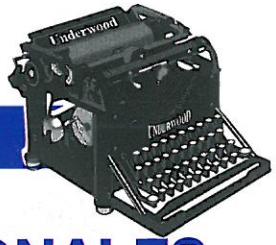
**Impresión**
Gráficas Reunidas, S. A.

**Distribución**
SGEL

**La revista SÓLO PROGRAMADORES** no tiene por qué estar de acuerdo con las opiniones escritas por sus colaboradores en los artículos firmados.
El editor prohibe expresamente la reproducción total o parcial de los contenidos de la revista sin su autorización escrita.

**Depósito legal:** M-26827-1994
**ISSN:** 1134-4792

# EDITORIAL




## LOS BUENOS PROFESIONALES

Como mencionaba hace un par de números en la habitual sección de Web en Sólo Programadores, habrá que continuar expectantes ante la evolución del mercado y es que, guste o no, todo se resuelve en términos de mercado. Ya en los primeros artículos de la serie se apuntaba como causa de la rápida evolución del WWW, su simplicidad. Posteriores ampliaciones de HTML hicieron este lenguaje más complejo, pero aún accesible a la gran mayoría del público. La interfaz CGI se hacía imprescindible para satisfacer ciertas necesidades básicas y *reales*. En cualquier caso, con todo ello se potenciaba la actividad comercial al convertir la red en un buen escaparate de marketing.

Ahora, es verdad, Java está de moda. Por el momento, la tendencia es "apuntarse" a Java, por el mero hecho de "estar a la última" o "incorporarse las nuevas tecnologías". Es decir, se toma Java como un fin en sí mismo y su utilización proporciona "imagen". Pero en la práctica, raras veces se justifica para el WWW. Se puede caer en el típico error de pensar en la técnica como un fin y no como un medio para mejorar la calidad de vida. Es cierto que Java ofrece un sinfín de posibilidades para el WWW, pero como toda tecnología no es intrínsecamente buena ni mala, depende de cómo se utilice.

Con Java, el WWW pierde su sencillez y ya no es accesible a cualquiera como desarrollador. En contadas ocasiones es necesario desarrollar algo tan complejo que requiera emplear Java. Aunque en estos casos no sólo es la única sino, además, una muy buena solución. En definitiva, cuando pase la marea, quizá Java se quede en su sitio (siempre hablando del mundo WWW) como herramienta para profesionales y se emplee únicamente cuando sea adecuado. Por el momento, las páginas de Internet se llenan de applets tan vistosos como inútiles pero que aumentan el tiempo de carga de las páginas. A veces, incluso, uno se pierde "jugando" con los applets y no llega a fijarse en el producto que se le pretendía vender, en la idea que se le trataba de transmitir, o no accede a la información buscada tan rápido como hubiera deseado.

Éste no es un vicio que surja únicamente con Java. Un buen desarrollador de páginas WWW debe saber vender bien un producto, dejar clara una idea. En definitiva, mostrar lo que quiere mostrar. Y no debe distraer al usuario, invitado o cliente con un fondo, un applet o unas animaciones estupendas que le aparten de la idea que se desea transmitir. Numerosas "empresas" que presumen de facilitar asesoría y desarrollo de páginas WWW realizan vistosos servidores que muestran a sus clientes leyendo de su disco duro o de la red local y por los se pagan considerables sumas de dinero. Pero la cruda realidad es que, a la hora de la verdad, a esas páginas se accederá desde la red, no sólo con un ancho de banda limitado sino, además con quien sabe qué problemas de latencia o QOS (Quality Of Service). Muchos de estos servidores jamás son visitados porque el usuario se aburre con los tiempos de carga de sus páginas.

Por tanto, un buen profesional no se debe limitar al aspecto técnico, debe buscar la solución más adecuada para el fin que se trata de conseguir. Cualquier cliente que se juegue su dinero sabrá apreciar una decisión responsable aunque proporcione resultados menos vistosos. Así, a la hora de colocar una página en Internet, conviene saber en que red se encontrará y por dónde, no sólo desde dónde, se espera que accedan los potenciales visitantes. Por ello, en el peor de los casos, en que no se pueda prever el origen del acceso o las condiciones de la comunicación, puede no ser mala idea ofrecer dos caminos diferentes al usuario para recorrer el servidor: uno para condiciones óptimas y otro para condiciones "menos óptimas", idea que se está ya empleando con frecuencia en Internet.

En la medida en que el desarrollador, ya sea como empresa o como particular, sienta la necesidad de demostrar sus conocimientos técnicos, quizá sea una mejor idea disponer de una zona de "demos" en lugar de ralentizar su trabajo con un *overhead* de accesorios inútiles.

El objetivo que persigue el autor con sus artículos es ayudar al lector a ser un buen tecnólogo y no simplemente un buen técnico. En definitiva, la labor de un buen profesional se puede resumir en una sola frase: "aportar soluciones".

Fernando Echevarrieta

# SUMARIO


6 **NOTICIAS:**
## LO ÚLTIMO EN INFORMÁTICA

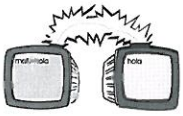
11 **TECNOLOGÍA WEB (IX):**
## EMPLEO DE FRAMES
En este artículo se presenta un tema que ha cuajado recientemente en la red: la división de ventanas a través del empleo de *frames*.

17 **CURSO DE PROGRAMACIÓN (I):**
## LA CREACIÓN DE UNIDADES
Es una de las partes más importantes de un lenguaje, ya que permite utilizar una misma función en distintos programas. Aquí explicamos cómo crearlas.

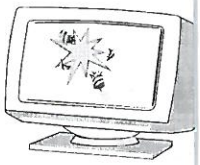
22 **CURSO DE UNIX (XIII):**
## PLAN 9
Plan 9 es el resultado de la cooperación de un grupo de gente entre los que se encuentran Ken Thompson, Dennis Ritchie y Robert Pike.

28 **GRANDES SISTEMAS (XIV):**
## PROGRAMACIÓN EN NATURAL (II)
Segunda parte del tema dedicado a los Objetos en NATURAL, donde se profundiza en el uso que se debe hacer de ellos.

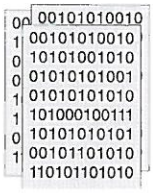
33 **REDES LOCALES (IV):**
## INTERCONEXIÓN DE REDES (II)
A veces no basta con un dispositivo que interconecte físicamente la red: se necesita un dispositivo inteligente.



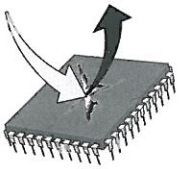
38 **CURSO DE DEMOS (XIV):**
## ALGORITMOS FRACTALES
En esta entrega se estudiarán los algoritmos fractales, que son utilizados cada vez con más frecuencia por todos los seguidores de la *demoscene*.



41 **PC INTERNO (I):**
## LA BIOS (I)
La BIOS es una de las partes más importantes del sistema y sobre ella funcionan casi todos los programas realizados hasta ahora en DOS.




46 **CREACIÓN DE UN COMPILADOR (X):**
## FIN DEL COMPILADOR
Se termina el compilador. Los programas fuente en Letra se convierten ya en programas ejecutables.



52 **MODOS X DE LA VGA (III):**

## EL CRTC CONTROLLER

En esta ocasión se tratará el uso del *CRCT Controller*, muy importante para poder controlar el refresco de la pantalla a través de sus registros.

56 **COMUNICACIONES SERIE:**

## COMUNICACIÓN ENTRE ORDENADORES

La comunicación entre ordenadores es un tema muy extenso y complejo de estudiar. En este artículo se tratará la manera de transmitir ficheros.

61 **FORMATOS DE SONIDO (III):**

## FICHEROS WAV

Se continúa la serie sobre formatos de ficheros de sonido con el formato para sonidos digitalizados más extendido en la actualidad: el formato.

64 **SISTEMA OPERATIVO LINUX (XVII):**

## EL RAMDISK DE LINUX

Tras abordar el mes pasado el uso de un disco de arranque mediante un disco RAM, ahora se estudiará su implementación en el núcleo de Linux.

69 **VISUAL C++ 4.0 (VII):**

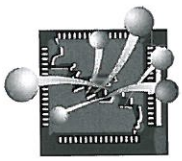
## HERRAMIENTAS DE DEPURACIÓN

En esta entrega del, como son las herramientas de depuración de errores de aplicaciones que ofrece Visual C++ 4.0.

74 **CURSO DE VISUAL BASIC 4.0 (VI):**

## LOS CONTROLES BÁSICOS (II)

Continuación de la parte dedicada a estudiar los Controles Básicos que ofrece Visual Basic para la creación de aplicaciones.



79 **CONTENIDO DEL CD-ROM:**

## TCL/TK EJEMPLOS Y DOCUMENTACIÓN

El TCL, *Tool Command Language*, es básicamente otra shell más del sistema operativo Unix, aunque cuenta con una semántica propia.



81 **CORREO DEL LECTOR:**

## CONSULTAS DE LOS LECTORES

Espacio destinado a resolver los problemas surgidos a los lectores en diversos aspectos de la programación.

## COMUNICACIÓN PUERTO SERIE

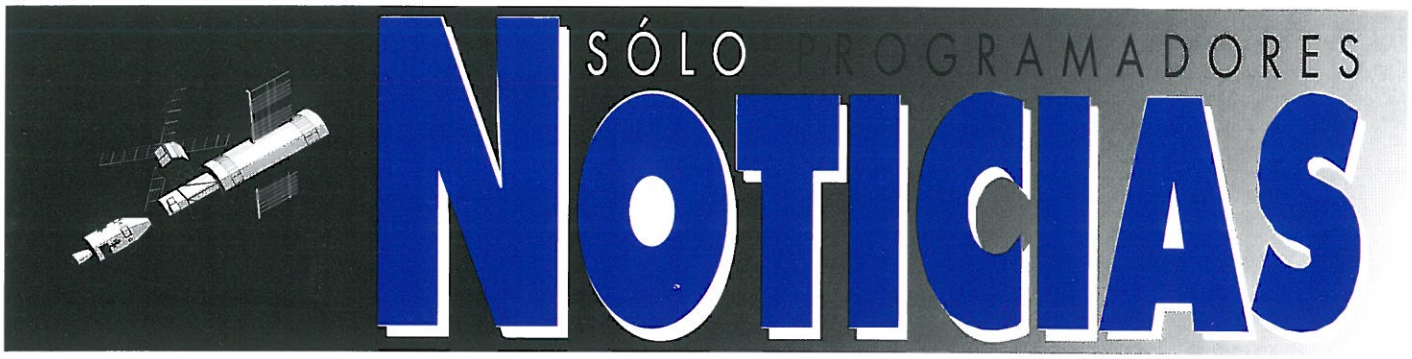
### Programación del Puerto Serie

Viene a nuestra portada este mes el artículo dedicado a las comunicaciones por el puerto serie, es posible que el tema de la comunicación entre ordenadores sea uno de los más extensos y complejos del mundo de la informática. En nuestro estudio se expone, desde el principio, la manera de transmitir ficheros entre dos ordenadores a través del puerto serie.

## CREACIÓN DE UN COMPILADOR

### Fin del Compilador

Especialmente es destacable en este número la última entrega de Creación de un Compilador, curso comenzado ya hace bastante meses, ahora en Agosto se termina el compilador, los programas fuente en Letra se convierten ya en programas ejecutables. También queremos destacar este mes dentro de Tecnología Web el artículo dedicado a el empleo de Frames.




# Natural Image

## SOFTWARE AG anuncia su nueva herramienta de proceso de imágenes en color

Basada en el lenguaje de programación 4GL Natural de Software AG, esta nueva herramienta permite por primera vez la integración de imágenes, incluyendo fotografías en color, en los documentos y archivos de aplicaciones corporativas.

Natural Image permite el proceso de imágenes en color y su integración en estaciones de trabajo con interface gráfico, y ofrece un rápido acceso SQL a las imágenes almacenadas en bases de datos relacionales. Además esta herramienta soporta el hardware y los entornos de software más comunes del mercado, incluyendo Pcs, Unix y Mainframe.

Natural Image PC reside en la estación de trabajo del cliente, y recibe y ejecuta las demandas de imágenes. Estas peticiones incluyen la captura, rotación, ampliación, display de imágenes y su impresión.

Para más información:
Software AG España
Ana Jerez
Tlf: (91) 807 96 95
Fax: (91) 807 94 70

# NAVEGADOR HISPANO

## Intercom presenta la manera fácil de navegar por Internet

Con el navegador Hispano, Intercom crea una guía de fácil uso de Internet, que permite a los usuarios españoles, de América Latina y Centroamérica disponer de forma organizada de todas aquellas informaciones que aparezcan en la Red en lengua castellana u otras lenguas ibéricas (catalán, gallego, vasco, etc.) para determinar así de forma ágil sus preferencias.

El Español, como corresponde con su papel de tercer idioma mundial, goza de muy buena salud en Internet. Poner toda esta información al alcance de los usuarios hispano parlantes ha sido uno de los objetivos impulsados por Intercom Servicios Telématicos Avanzados a crear el Navegador Hispano.

Esta herramienta agrupa todas las informaciones en cualquiera de sus idiomas o lugares de preferencia, organizándolas en tres secciones. Estas son: "Navegación", "Buscadores" y "Selección de páginas".

En "Navegación", se ofrece a los usuarios más de 2000 recursos distintos agrupados de forma temática. En "Buscadores" se ofrece accesos a los buscadores más clásicos y reconocidos como Yahoo, Alta Vista y Lycos.

# OCR FOR FORMS

## Extracción automática de datos

Microsystems Technologies ha presentado en España Ocr for FORMS, una aplicación que permite la extracción automática de los datos contenidos en formularios conocidos. Ocr for Forms extrae y suministra los datos de los documentos apoyándose en técnicas de reconocimiento óptico de caracteres, marcas y caracteres manuscritos, evitando el coste que significa su extracción manual. El producto está diseñado para funcionar en red y dispone de unos algoritmos de tratamiento de la imagen que permiten la eliminación de la imagen, el control sobre el OCR/OMR/IMR a realizar sobre los datos obtenidos mediante tablas o filtros, y la salida ASCII de los datos.

# CRECIMIENTO DE COBOL

## Los lenguajes COBOL experimentan un crecimiento sostenido desde 1993

Un informe de la consultora americana Sentry Market Research señala que los lenguajes COBOL y COBOL Orientado a Objetos (OCC)están experimentando un crecimiento sostenido desde el citado año en entronos cliente/servidor. Dicho informe indica que estos lenguajes han escalado posiciones hasta ocupar el segundo lugar en el ranking de preferencias de los usuarios. Así mismo, el COBOL Orientado a Objetos también fue votado como el segundo mejor, desbancando incluso a Visual C++, Visual Basic y Small Talk.

# INTERNET EXPLORER 3.0

## Ya está disponible la versión beta

Microsoft Corporation ha anunciado la preparación de la próxima versión de su navegador Internet Explorer. Este es el primer programa cliente para Internet que integra las tecnologías ActiveX, que permiten a los desarrolladores crear aplicaciones y contenido interactivo multimedia para Internet. Estas tecnologías permitirán que una dirección web sea tan interactiva y completa como un juego de acción, una enciclopedia multimedia o una aplicación de productividad.

Internet Explorer 3.0 es compatible con las últimas especificaciones HTML del World Wide Web Consortium (W3C), así como con otros estándares populares como Java, JavaScript, Visual Basic y VB Script.

Con más de 1000 socios comprometidos con la tecnología ActiveX, el lanzamiento de la versión 3.0 es un elemento fundamental en la publicación de contenidos de vanguardia.

Las tecnologías Activex estarán disponibles en varias plataformas, con lo que tanto los usuarios como los desarrolladores contarán con una estructura amplia e innovadora para aumentar sus *conocimientos* e *inversiones* actuales en herramientas, aplicaciones y código fuente.

Para más información:
http://www.microsoft.com

# TERMINALES PORTÁTILES

## Nuevos terminales portátiles de Epson para aplicaciones profesionales

Estos nuevos terminales, modelos EHT-40/30/30L, facilitan la gestíon de datos o información en cualquier lugar de trabajo. El modelo EHT-40 incorpora un potente procesador compatible 386sx, que permite hasta 14 horas de operación. Una extensa gama de opciones permite configuar la serie EHT para todo tipo de aplicación.

Estos terminales son ideales para aplicaciones como la automatización de factorías, vendedores, toma de pedidos, gestión de almacen, lectura de contadores para compañías de agua, luz, gas, etc. El uso de un lector de código de barras permite un rápido control de inventario, que mediante la base de comunicaciones puede ser enviado a un ordenador central. Existen tres impresoras disponibles para la serie EHT aptas para cualquier aplicación. También se puede incorporar lector de tarjetas magnéticas.

Para más información:
Epson Ibérica
Tlf: 900 486 000
fax: (93) 582 15 55

# C PARA APLICACIONES INTRANET/INTERNET

## Attachmate presenta nuevas herramientas de desarrollo basadas en ActiveX

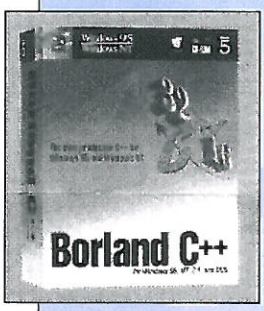
Attachmate Corporation, empresa líder en software de conectividad y construcción de Intranets, ha anunciado la incorporación a su familia de productos Extra! Developer Serie de Emmisary Host Publishing System. Este producto permite a las compañías desplegar aplicaciones Web realizadas con las herramientas de desarrollo cliente/servidor QuickApp y QuickDB, que ofrecen acceso a datos host de una manera sencilla y fiable al usuario final.

Este conjunto de herramientas incluirá soporte ActiveX que convierte automáticamente la aplicación en formato HTML, y eliminará la necesidad de formar a los programadores cliente/servidor acerca de las complejidades del desarrollo de Web.

# NOVEDADES

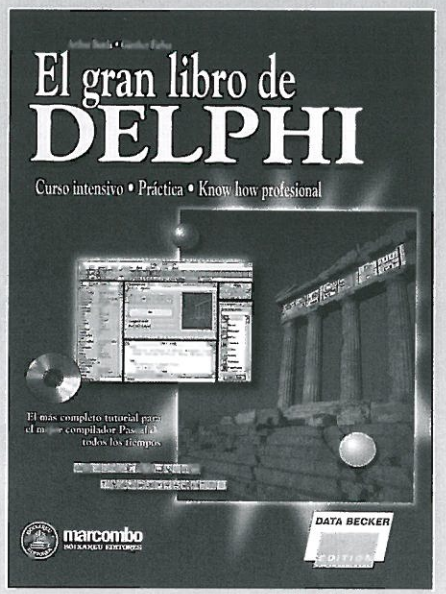
## Nueva versión de Borland C++ para Windows 95 y NT con compilador Java

Borland, una de las empresas líderes en desarrollo de lenguajes de programación, ha lanzado la nueva versión 5.0 de su popular C++. Con un nuevo y potente Entorno de Desarrollo programado sobre sí mismo, ofrece a los programadores de C++ una productividad extraordinaria, unida a la potencia y velocidad de uno de los compiladores más rápidos del mercado. Además, Borland C++ 5.0 ofrece lo último en tecnología multiplataforma al incluir JAVA, el lenguaje de programación más robusto y compacto para desarrollar aplicaciones para Internet.




Entre las nuevas herramientas incluidas en este paquete destaca un nuevo soporte para MFC, una nueva librería estándar C++, nuevo depurador de 32 bits integrado y nuevas herramientas de programación Java. Así mismo, ofrece simultáneamente un IDE de 32 Bits y otro de 16 Bits, con lo que los programadores tendrán ahora la opción de elegir tanto el entorno en el que van a trabajar como el entorno para el que van a compilar, obteniendo en todos los casos la misma implementación de C++, independientemente de la plataforma de destino. La actualización a esta nueva herramienta se encuentra disponible al precio promocional de 24.900 pesetas en su versión básica o 34.900 para la Development Suite. Esta oferta es válida hasta el 31 de agosto.

# LIBROS



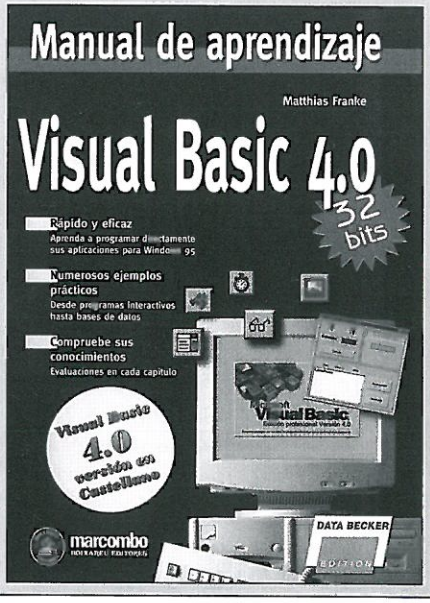

## El gran libro de Delphi

Delphi es una herramienta de programación creada para el desarrollo de aplicaciones Windows con lenguaje Pascal. El gran libro de Delphi es una obra dirigida principalmente a programadores inexpertos que enseña los secretos de la programación con Delphi empezando desde el principio. La obra comienza con una pequeña introducción a Delphi para entrar de lleno en un curso de iniciación que proporciona los conocimientos básicos necesarios para el desarrollo de programas con esta herramienta. Posteriormente se tratan, de forma progresiva, los aspectos más avanzados. En todos los capítulos se acompaña el texto con diversos gráficos y diagramas destinados a clarificar los conceptos. Además, el libro incluye un CD-ROM con todos los programas de ejemplo.

Editorial: Marcombo
Autores: Arthur Burda y Günther Färber
438 páginas
Idioma: Castellano
Precio: 4.900 (IVA incluido)

## Visual Basic 4.0. Manual de aprendizaje

Se trata una obra ideal para programadores autodidactas dedicada por entero al desarrollo de aplicaciones con la última versión de Microsoft Visual Basic. Este manual de aprendizaje está estructurado en cuatro partes en las que se introduce al lector en las características del producto. La primera parte es una breve introducción que explica, entre otras cosas, la forma de instalar los programas del disco adjunto. La segunda parte trata los fundamentos de la programación con Visual Basic, mientras que la tercera incluye temas avanzados como la búsqueda de errores y el almacenamiento de datos. La última parte está dedicada a técnicas avanzadas como los formularios MDI y el acceso a bases de datos.




Editorial: Marcombo
Autor: Matthias Franke
455 páginas
Idioma: Castellano
Precio: 4.900 (IVA incluido)

# ENCUESTA SÓLO PROGRAMADORES

## RELLENA ESTA ENCUESTA Y PARTICIPA EN UN ESTUPENDO SORTEO

| SECCIONES | VALÓRALAS DE 1 A 10 |
|---|---|
| NOTICIAS Y LIBROS | 1 2 3 4 5 **6** 7 8 9 10 |
| CARTAS DEL ILUSO | 1 2 3 4 5 6 7 8 **9** 10 |
| TECNOLOGÍA WEB | 1 2 3 4 5 6 7 **8** 9 10 |
| CURSO DE PROGRAMACIÓN | 1 2 3 4 5 6 7 **8** 9 10 |
| SISTEMAS ABIERTOS | 1 2 3 4 5 6 **7** 8 9 10 |
| GRANDES SISTEMAS | 1 2 3 4 5 6 **7** 8 9 10 |
| CURSO DE REDES | 1 2 3 4 5 6 7 **8** 9 10 |
| SIST. OPERATIVO LINUX | 1 2 3 4 5 6 7 **8** 9 10 |
| CÓMO PROG. UNA DEMO | 1 2 3 **4** 5 6 7 8 9 10 |
| CREACIÓN DE UN COMPILADOR. | 1 2 3 **4** 5 6 7 8 9 10 |
| PROGRAMACIÓN DE LA VGA | 1 2 3 4 **5** 6 7 8 9 10 |
| FORMATOS DE SONIDO | 1 2 3 4 **5** 6 7 8 9 10 |
| CURSO VISUAL BASIC | 1 2 3 4 5 6 7 8 **9** 10 |
| CURSO VISUAL C++ | 1 2 3 4 5 6 **7** 8 9 10 |

¿Qué te parece el nivel de los artículos?

- ☐ Muy alto
- ☒ Alto
- ☐ Normal
- ☐ Bajo
- ☐ .....

¿Qué te parece en general el contenido de la revista?

- ☐ Muy bueno
- ☒ Bueno
- ☐ Regular
- ☐ Malo
- ☐ .....

¿Qué cambios te gustaría realizar? ¿Cúales?

- ☒ AÑADIR NUEVAS SECCIONES — Curso C, Bases Datos
- ☒ QUITAR SECCIONES — DEMO, COMPILADOR
- ☒ AMPLIAR LAS ACTUALES — Redes, UB, Programac.

¿Te conectas a algún servicio On-Line?

- ☐ Sí
- ☐ No
- ☐ Habitualmente
- ☒ Esporádicamente

¿Cúal?

- ☐ INTERNET
- ☐ INFOVÍA
- ☐ COMPUSERVE
- ☒ BBS
- ☐ OTROS

### DATOS PERSONALES

Nombre y Apellidos PEDRO SALAZAR ROMAN

Domicilio C/ MELCHOR FDEZ. ALMAGRO, 35

Población MADRID

C.P. 28029

Provincia MADRID

Teléfono 3145025

Edad 26

Estudios UNIVERSITARIOS

Profesión PROFESOR

Ordenador de que dispone PENTIUM

Configuración de tu equipo:

| | | | | | | |
|---|---|---|---|---|---|---|
| Procesador | 386 | 486 | ~~PENTIUM < 120~~ | PENTIUM > 120 | | |
| Tarjeta Gráfica | VGA | ~~SVGA~~ | 65K COLORES | 16,7M COLORES | | |
| Tarjeta de Sonido | SOUND BLASTER | ~~SOUND BLASTER 16~~ | GRAVIS ULTRA | | | |
| CD-ROM | 1X | 2X | ~~4X~~ | 6X | 8X | |
| Módem | 2.400 | 9.600 | ~~14.400~~ | 28.800 | | |

RELLENA EL CUESTIONARIO Y ENVÍALO A:

**SÓLO PROGRAMADORES ( ENCUESTA)**
**C/ Aragoneses, 7 - 28100 Pol. Ind. Alcobendas (Madrid)**

**Entre todas las encuestas recibidas antes del día 1 de Octubre se sortearán**

- **6 modems internos de 14.400 baudios**
- **1 BORLAND DELPHI DEVELOPER y 1 BORLAND C++ para Windows 95 y NT**

# ACTIVEX: LA EVOLUCIÓN HACIA LOS CONTENIDOS ACTIVOS EN INTERNET (I)

El mundo Internet y de las intranets corporativas evoluciona a un ritmo muy rápido. Cada vez son más los usuarios y las empresas que han elegido este nuevo vehículo de comunicación virtual, para compartir experiencias, comercializar productos, acceder y ofrecer información. Microsoft como empresa pionera en tecnología y comprometida con los usuarios trabaja continuamente para ofrecer nuevas soluciones y tecnologías para Internet a los usuarios.

Ante la rápida evolución de este nuevo mundo, Microsoft desarrolló las tecnologías ActiveX, inaugurando así una nueva etapa de contenidos activos en Internet. Estas tecnologías han sido diseñadas para facilitar a los desarrolladores de software y de la web la construcción de contenido activo para Internet y el PC. Con las tecnologías ActiveX, las páginas web pasan de ser estáticas a incluir contenido "vivo" como animaciones, realidad virtual 3D, vídeo y otros elementos multimedia. Concretamente, las tecnologías ActiveX permiten a los desarrolladores crear fácilmente contenido web utilizando los Controles ActiveX (controles tipo OLE), programación activa y documentos activos, al mismo tiempo que permite la utilización de los componentes de software, la programación y las aplicaciones existentes.

Los Controles ActiveX son pequeños componentes, pero con todas las propiedades para Internet, Intranet y PCs. Los Controles ActiveX cuentan ya con un amplio apoyo de cientos de fabricantes de software; representan el soporte necesario de contenido activo. Los Controles ActiveX funcionan con numerosos lenguajes tanto de Microsoft como de otras empresas, incluyendo Microsoft Visual Basic, Microsoft Visual C++, Borland Delphi y la futura herramienta de programación de Microsoft para Java, denominada Visual J++ (Jakarta).

Los Controles ActiveX permiten a los desarrolladores integrar directamente en páginas HTML una amplia variedad de componentes de software, como visores de gráficos, animaciones, objetos para transacciones con tarjetas de crédito u hojas de cálculo. Por ejemplo, ActiveMovie, la tecnología de sonido y reproducción de vídeo, es un Control ActiveX que permite a los usuarios de Internet Explorer 3.0 ver imágenes de vídeo directamente en su explorador. Gracias a la disponibilidad inmediata de cientos de Controles ActiveX, los desarrolladores pueden empezar ya a crear contenido activo para Internet. Además, los applets Java pueden compartir espacio con los controles ActiveX dentro de una misma página HTML.

Una ventaja clave de las tecnologías ActiveX es la posibilidad de integrar las aplicaciones en los exploradores de la web para que los datos empleados en las aplicaciones sean tan accesibles como las mismas páginas web. Esta tecnología, llamada Document Objects o de documentos activos, permite a los usuarios navegar por una red corporativa o intranet para ver una página web de un departamento, examinar la hoja de cálculo con el presupuesto de un departamento, consultar la base de datos del departamento de ventas o escribir una nota, todo desde el explorador para web y sin la necesidad de realizar la conversión a formato HTML.

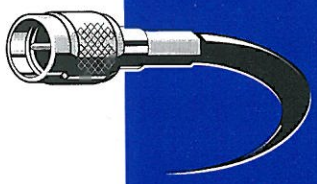
Otro componente de las tecnologías ActiveX es ActiveX Server Framework. Este entorno de trabajo, basado en Microsoft Internet Information Server integrado con Windows NT Server, permite a los desarrolladores crear las mismas aplicaciones interactivas para el servidor, utilizando la experiencia, conocimiento y herramientas existentes. Permite a los desarrolladores para la web aumentar las posibilidades de la familia Microsoft BackOffice, incluidos los actuales Microsoft SQL Server, Microsoft SNA Server, Microsoft Systems Management Server y Microsoft Exchange Server.

ActiveX Server Framework se compone del código ActiveX Server y de los controles ActiveX Server. Los controles ActiveX Server son los elementos principales del servidor para proporcionar contenido activo, permitiendo a los usuarios aprovechar los sistemas heredados o construir rápidamente aplicaciones con componentes de objetos reutilizables. El código ActiveX Server puede escribirse utilizando un abanico de lenguajes de programación conocidos, como Visual Basic Script, Perl y JavaScript. En conjunto, el entorno de trabajo facilita la posibilidad de integrar el web con los entornos y las aplicaciones de misión crítica que los usuarios poseen actualmente, permitiendo, por ejemplo, orientar las consultas de bases de datos y acceder a los mainframes. Actualmente, más de 50 fabricantes de software independientes apoyan el entorno de trabajo del servidor ActiveX y cientos de usuarios han comenzado a crear direcciones web activas basadas en este entorno de trabajo.

**José Antonio Alvarez**
**Jefe de Producto de Herramientas de Desarrollo de Microsoft Ibérica**

# LAS VENTANAS LLEGAN AL WWW

*Fernando J. Echevarrieta*



Como sabe ya todo aquel lector que haya venido siguiendo la serie, las tecnologías que rodean la gran tela de araña que se ha desarrollado sobre internet son uno de los terrenos de más rápida evolución. Cuando hace apenas unos artículos que en la revista se anunciaba una nueva propuesta de Netscape denominada *frame*, ya se ha convertido en una de las prácticas de moda en todo servidor "que se precie" y es que, incluso, se hace abuso de ella fragmentando a diestro y siniestro las ventanas de los browsers. Y aunque, una vez más, no se trata de una extensión estándar, sino de una propuesta de Netscape, se ha convertido en un estándar de facto ante el que se ha rendido incluso Microsoft con su Internet Explorer. Por ello, este mes se dejan a la espera otras tecnologías más complejas para abordar, en profundidad, ésta, que no por más sencilla y accesible deja de ser de gran importancia.

## INTRODUCCIÓN A LOS FRAMES

Las versiones beta de Netscape Navigator 2.0 soprendieron al público por una nueva propuesta denominada *frame* (*cuadro*). Mediante el uso de frames era posible dividir una página de WWW en varias zonas independientes o frames, cada uno de los cuales podía ser explorado también de forma independiente, apuntar a un URL diferente o estar dotado de scroll también por separado. La existencia de frames se hacía posible a través de nuevas extensiones de HTML, por lo que la compatibilidad con otros browsers, como en ocasiones anteriores, quedaba asegurada. Desde entonces, el crecimiento del número de servidores que emplean esta técnica ha sido exponencial: al principio, los servidores adoptaron tímidamente esta nueva extensión, apenas unos pocos que figuraban en la lista del WWW de Netscape; en estos momentos, ya es habitual encontrar frames en cualquier lugar de la red. De hecho otros browsers como Microsoft Internet Explorer ya han incorporado esta nueva idea.

Entre las ventajas que proporcionan los frames se encuentran las posibilidades de presentar en una zona de la pantalla los resultados de ejecutar una acción en otra. Así, por ejemplo, es habitual la presencia de un *frame* a modo de menú de acceso rápido a puntos clave de un servidor. Este *frame* permanece siempre visible y al pulsar sobre una de sus opciones, se accede a la misma en otro *frame* que representa la ventana principal.

Otra de las posibilidades que facilitan es la disponibilidad permanente de una zona de la pantalla a la que se puede fijar el tamaño, independientemente de si el usuario debe hacer scroll de la ventana principal de navegación o cambiar de URL durante su recorrido. Así, se suelen presentar logotipos, encabezados, pies de página o botoneras en forma de frames que permanecen fijos en pantalla.

La extensión *frame* está dotada de una gran flexibilidad. Así, cada *frame* dispone de un URL individual, lo que permite, en una sóla página de WWW emular el comportamiento de varios browsers funcionando a la vez, pero según la idea de presentación concebida por el proveedor de la información. Con la finalidad de permitir reflejar en un *frame* concreto determinadas acciones, se ofrece también la posibilidad de dotar de nombre a cada una de estas ventanas. Por último, es posible realizar páginas con frames de tamaño fijo, dinámi-

**Hace unos meses se comentaba como noticia la aparición de una nueva extensión HTML que ofrecía un nuevo aspecto de división en ventanas a los documentos de WWW: el empleo de frames. La idea, que al principio fue aceptada con timidez, ha cuajado ya en la red, por lo que este artículo desvela todo lo necesario para su creación y manejo**

camente redefinible (si el usuario cambia el tamaño de su ventana), de tamaño redimensionable por el usuario (con bordes deslizantes) e, independientemente de lo anterior, que permitan o no realizar scroll.

Por el momento, los frames son una propuesta enviada oficialmente al *IETF* (*Internet Engineering Task Force*) y al *World Wide Web Consortium* para su consideración como estándares de internet. Aunque aún no se tiene respuesta, la opinión del autor, como se justifica

**TABLA**

```
<FRAMESET [ROWS=listaICOLS=lista]>
<FRAME SRC= NAME= MARGINWIDTH= MARGINHEIGHT= SCROLLING= NORESIZE=>
<NOFRAMES>
```

**Etiquetas correspondientes al uso de frames.**

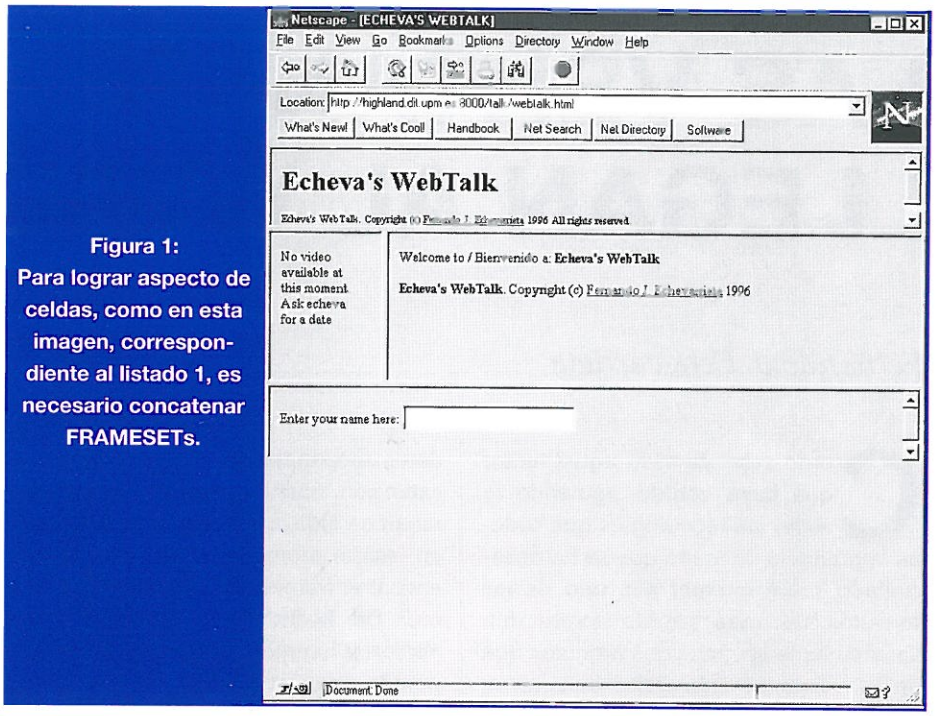



**Figura 1: Para lograr aspecto de celdas, como en esta imagen, correspondiente al listado 1, es necesario concatenar FRAMESETs.**

más adelante, es que no deberían ser aceptados con la implementación actual. Si bien la interfaz presentada a los usuarios es una idea que habría que recoger, la sintaxis de HTML definida debería ser cambiada.

## DOCUMENTOS FRAME

La nueva extensión de frames incorpora cuatro nuevos elementos: un nuevo tipo de documento: *frame*, y dos nuevas etiquetas HTML: *<FRAMESET>*, *<FRAME>* y *<NOFRAMES>*.

Un documento de tipo *frame* es aquel que se encuentra dividido en frames y dispone de una estructura análoga a la de cualquier documento HTML, salvo por que, en este caso, no puede aparecer la etiqueta doble *<BODY>*, que ha de ser sustituida por la nueva etiqueta doble *<FRAMESET>*. Así, la estructura básica queda definida de la forma siguiente:

```
<HTML>

<HEAD>
</HEAD>

<FRAMESET>
</FRAMESET>

</HTML>
```

Esta es la razón por la que, en opinión del autor, no se debería aceptar esta extensión. Si bien en todo momento se ha tratado de expresar, con mayor o menor acierto (y a veces con algunas inconsistencias), el HTML como DTD de SGML, ahora será más difícil al definir ciertos documentos que carecen de "cuerpo". El motivo de esto es que el documento principal contendrá una serie de frames con su propio URL que apuntará a verdaderos documentos con cabeza y cuerpo.

## DEFINICIÓN DE FRAMES

Para definir la estructura de los frames de un documento se emplea la etiqueta *<FRAMESET>*. Si bien la documentación en línea de Netscape presenta como análogas las sintaxis empleadas en frames y tablas, el uso de los primeros es mucho más sencillo. La etiqueta *<FRAMESET>* dispone únicamente de dos posibles atributos: *ROWS* y *COLS*. La idea fundamental a la hora de realizar documentos con frames es pensar en dividirlos en conjuntos de frames (*FRAMESETs*) horizontales o verticales, pero no las dos cosas a la vez. Con este planteamiento se evita el engorro que aparecía en la definición de tablas cuando se realizaban celdas que cubrían más de una fila o columna o ambas cosas a la vez.

*ROWS*: Consiste en una lista de valores separados por comas. El documento será dividido en tantos *frames* como valores se indiquen y se les dimensionará conforme a estos valores. Los valores pueden ser expresados en número absoluto de pixels, valor porcentual o valor relativo de escalado. Dado que la suma total de las dimensiones de cada *frame* debe ser la dimensión total de la página, los valores que se indican en la etiqueta pueden ser normalizados. Los valores se indican de la siguiente forma:

*número*: Para indicar un valor absoluto en pixels.

*número%*: Para indicar un porcentaje entre 1 y 100. Si el total suma más de 100, es cuando el browser normalizará los valores para su correcta representación. Si suma menos de 100 y no existen *frames* de valor relativo, también se producirá una normalización, pero si existen (estos *frames* de valor relativo), se repartirá el espacio extra entre ellos.

*[número]**: Para indicar *frames* de valor relativo. Si no se indica número, es decir, si se proporciona un asterisco como tamaño del *frame*, se entiende que se le asignará el "espacio restante" de la página. En caso de que existan varios *frames* de tamaño relativo, se repartirá este espacio proporcionalmente al número indicado. Por ejemplo, "3*,*" significará una asignación del "espacio restante" de 3/4 partes al primer *frame* de tamaño relativo y 1/4 al segundo.

*COLS*: Actúa de forma análoga a *ROWS* pero definiendo, en este caso, la geometría de *frames* verticales.

Con el fin de dejar claros estos valores, se presentarán cinco ejemplos:

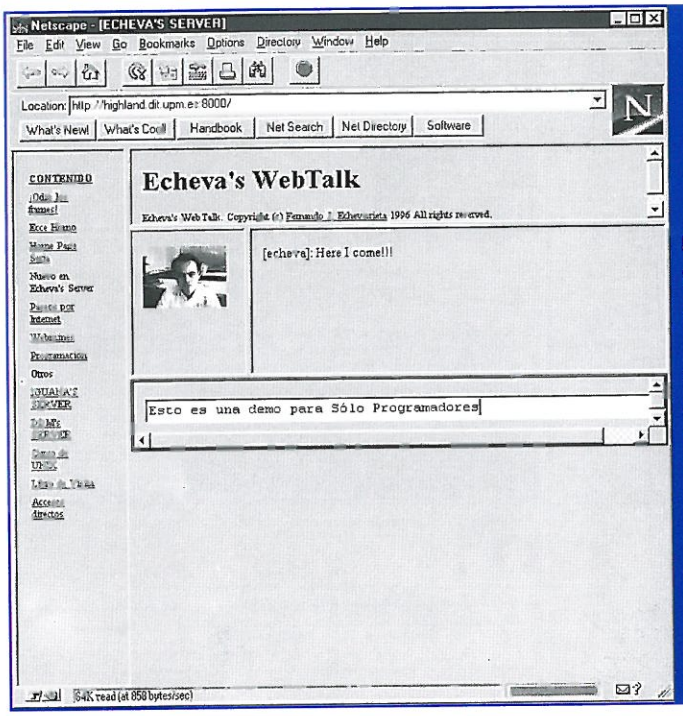



**Figura 2: Si navegando desde un frame se llega a un documento frame ambos se concatenan realizándose la división, no en la ventana completa, sino en el frame de navegación.**

*<FRAMESET ROWS="50%,30%,20%">*
Definiría un documento dividido en tres ventanas independientes, la primera de ellas ocupando el 50% del mismo, la segunda el 30% y la tercera, el 20%. En caso de que el usuario redimensionara la ventana de su navegador, también serían dimensionados los *frames* para adecuarse a los porcentajes.

*<FRAMESET ROWS="50%,30%,*">*
Produciría exactamente el mismo efecto ya que al tercer *frame* se le asignaría el "espacio restante", es decir, el 20%.

*<FRAMESET ROWS="75%,45%,30%">*
Definiría un documento en que la suma de los porcentajes ascendería a 150, por lo que el cliente normalizaría los valores a una escala de 100 presentando *frames* fila con una ocupación del 50%, 30% y 20%, respectivemante, exactamente como en los casos anteriores.

*<FRAMESET ROWS="50,30%,20%">*
Definiría un documento con un *frame* fila de 50 pixels y tamaño constante aunque el usuario redimensionara la ventana y otros dos *frames* de un 30% y un 20% por ciento, respectivamente, autoredimensionables.

*<FRAMESET ROWS="50,30%,2*,*">*
Generaría un documento *frame* con un primer *frame* fila de 50 pixels y tamaño fijo, un segundo *frame* que ocuparía un 30% del total de la ventana del navegador y otros dos *frames* que se repartirían el "espacio restante", llevándose el primero 2/3 del mismo y el segundo 1/3.

## DANDO CONTENIDO A UN FRAME

Una vez definida la estructura de los *frames* de un documento, se procede a configurar su contenido. Para ello, se emplea la etiqueta *<FRAME>*, cuya forma de uso más simple es *<FRAME SRC="URL">*. Así, el ejemplo más simple sería:

```
<FRAMESET ROWS="50%,50%">
  <FRAME SRC=ventana_1.html>
  <FRAME SRC=ventana_2.html>
</FRAMESET>
```

Dada la flexibilidad de comportamiento de los *frames*, ésta etiqueta admite hasta seis atributos:

*SRC*: Como ya se ha visto, indica el URL del documento que ha de ser mostrado en el *frame*. En caso de existir una etiqueta *<FRAME>* sin este atributo, se mostrará un espacio en blanco en su lugar.

*NAME*: Proporciona un nombre al *frame* con el objeto de que pueda ser apuntado por un enlace presente en otra ventana u otro *frame*. Es de este modo como se realizan los índices: al apuntar todos los enlaces de un índice a la ventana principal (posteriormente se verá cómo), el URL al que señalan es mostrado en ésta en lugar de sobreescribir el índice. Por defecto, todos los *frames* y ventanas que se generan carecen de nombre y como nombre puede emplearse cualquier cadena alfanumérica, salvo ciertas excepciones que se indicarán más adelante.

*MARGINWIDTH*: Especifica el valor en pixels de los márgenes horizontales entre los bordes de un *frame* y su contenido. Si no se especifica, o por ser demasiado amplio no permite que se presente el contenido, los decidirá el propio cliente.

*MARGINHEIGHT*: Funciona de forma análoga a *MARGINWIDTH* pero con los márgenes verticales.

*SCROLLING*: Especifica si deben o no aparecer barras de scroll horizontales y/o verticales en el *frame*. Puede tomar tres valores:

*yes*: Muestra siempre barras de scroll en el *frame*.

*no*: No muestra nunca barras de scroll en el *frame*.

*auto*: Deja que el browser decida si deben aparecer o no las barras en el *frame* y cuándo. Esta es la opción más utilizada, por lo que es la opción por defecto si no se especifica el atributo.

*NORESIZE*: En un documento con *frames*, se puede cambiar el tamaño de los mismos pulsando sobre sus bordes y arrastrándolos a una nueva posición. Si se especifica este atributo, que no acepta valores (se comporta como un flag), no se permitirá al usuario arrastrar y soltar sus bordes.

## ASEGURANDO COMPATIBILIDAD

Un documento *frame* no sólo no consta de cuerpo, sino que si en él figura alguna etiqueta que normalmente aparecería en el cuerpo de un documento HTML esto producirá que no funcione como un documento de *frames*. Sin embargo, dado que las nuevas etiquetas son extensiones, y si no pueden existir las de un cuerpo HTML, cualquier browser que no entienda de *frames* lo vería como un documento vacío. Para resolver esto, se añade la etiqueta *<NOFRAMES>...</NOFRAMES>*, que debe ser incluída dentro de *<FRAMESET>*. Así, un browser que entienda *frames* ignorará lo que figura rodeado por la etiqueta

<NOFRAMES> mientras que un browser que no sepa de *frames* encontrará en esta zona precisamente las etiquetas correspondientes al cuerpo del documento. Por ejemplo:

```
<FRAMESET COLS="20%,*">
<NOFRAMES>
Este documento se ve mejor con frames
(...)
</NOFRAMES>
<FRAME SRC=indice.html>
<FRAME SRC=bienvenida.html>
</FRAMESET>
```

Donde, en el espacio (...) se suele colocar, por ejemplo, el mismo contenido que en *indice.html*.

## CONCATENACIÓN DE FRAMES

Con esta forma de definir conjuntos de *frames*, sólo verticales o sólo horizontales, no se pierde generalidad, ya que es posible concatenar conjuntos de *frames* de la misma forma que en tablas en las que una celda podía estar constituída por una tabla entera.

Por poner un ejemplo, supóngase que se desea realizar el documento que aparece en la figura 1. En él se puede apreciar un primer *frame* con un título que definirá una ventana de navegación. En una segunda fila figuran, a la izquierda, una ventana destinada a video y a la derecha otra destinada a diálogo. Por último, una tercera fila muestra un campo de form y el resto del documento aparece vacío.

Como se mencionaba anteriormente, la organización de *frames* toma como elementos base únicamente filas o columnas. Así pues, el efecto de "casillas" debe ser logrado mediante concatenación de *framesets*. En la figura se puede ver que las zonas horizontales van de extremo a extremo del documento, no así las verticales. Así pues, la forma de razonarlo sería desarrollar tres *frames* horizontales: el primero y el último contendrían información apuntada por sendos URLs mientras que el segundo debería ser dividido en otros dos verticales, es decir, serviría para alojar un nuevo *frameset* concatenado.

El listado 1 muestra el código HTML asociado a la figura. En él se puede observar cómo para las filas primera y tercera, que constan de un solo *frame*, no se definen *framesets* (no sólo no es necesario sino que además no se debe hacer),

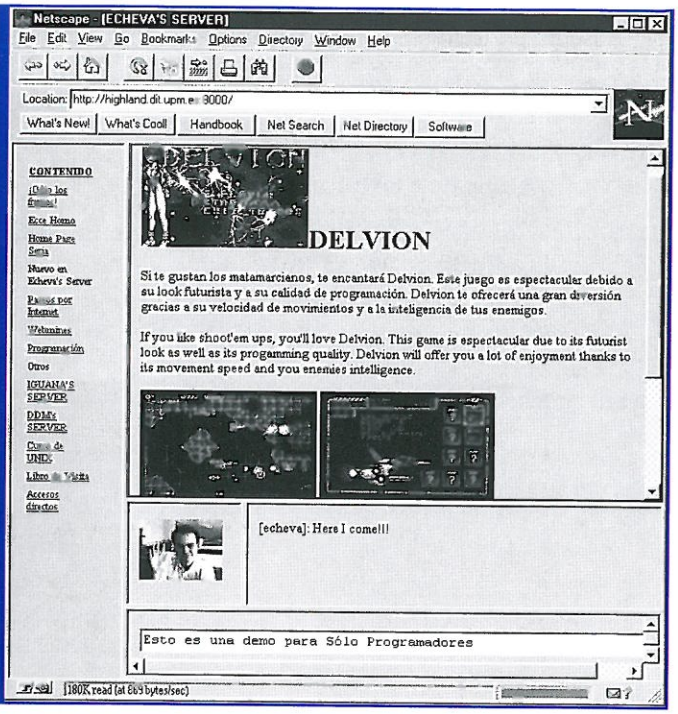



**Figura 3: Es posible realizar frames dimensionables por el usuario. En este ejemplo, el usuario ha ampliado el primer frame para navegar por el WWW al tiempo que continúa su charla en el resto de los frames.**

pero sí para la fila 2. Asimismo, se ha incluído una cuarta fila definida por una etiqueta <FRAME> sin atributo *SRC*, que como se ha expuesto y se corresponde con la figura, se traduce en un espacio en blanco. Esto se ha realizado con el objeto de poder deslizar los *frames* centrales hacia la parte inferior que al ser definida como un *frame* de tamaño relativo presenta un borde superior deslizable. De esta forma es posible navegar por la red en la primera ventana (tras darle un tamaño considerable), al tiempo que se lleva a cabo una conversación con los amigos en las otras, como se puede apreciar en la figura 3.

## APUNTANDO A OTRAS VENTANAS

Como se ha venido indicando, una de las primeras funcionalidades para las que se emplea un *frame* es para su utilización como índice. Pero para que esto tenga sentido, los enlaces que se pulsen en el *frame* índice deberán provocar la aparición del URL correspondiente en otro *frame* u otra ventana de visualización. Ya se ha visto que para identificar un *frame* se le da nombre mediante el atributo *NAME* de la etiqueta <FRAME>. Para referirse a un *frame* se emplea el nuevo atributo *TARGET* en la etiqueta <A> (como se verá más adelante, también podrá emplearse en otras etiquetas), por ejemplo:

```
<A HREF="bienvenida.html" TARGET="main">Bienvenida</A>
```

generaría un enlace con el texto *Bienvenida* que al ser pulsado provocaría la aparición del documento *bienvenida.html* en el *frame* o ventana que haya sido nombrado como *main*. En caso de no existir tal *frame* o ventana, el browser abriría una nueva ventana que recibiría el nombre de main para mostrar el documento. Esta es una particularidad que puede emplear el desarrollador de páginas HTML para abrir nuevas ventanas en el cliente sin necesidad de intervención del usuario.

En casos como el (ya desgastado en este artículo) ejemplo del *frame* índice, en que todos los enlaces dispongan del mismo *TARGET*, es posible omitirlos mediante la inclusión al principio del documento de la etiqueta <BASE>, a la que también se ha ampliado mediante la inclusión del atributo *TARGET*. Por ejemplo:

```
<BASE TARGET="main">
```

que provocará que cualquier enlace que no defina su propio atributo *TARGET*, que tendría preferencia, apunte al *frame* o ventana de nombre *main*.

Como se indicaba en la introducción, otro de los usos de los *frames* es ofrecer la posibilidad de mostrar en una ventana los datos introducidos en un *form* y en otra el resultado de procesarlos. Para ello, se extiende también la

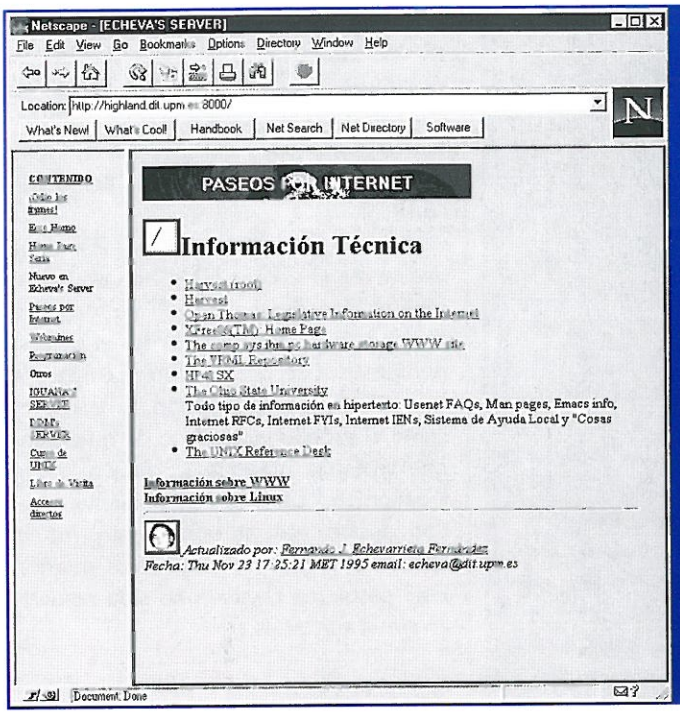



**Figura 4:**
**En casos como éste, es cuando se hace conveniente el uso de destinos base, como _top, y especiales, como _self.**

etiqueta *<FORM>* mediante el atributo *TARGET*. Por ejemplo:

*<FORM ACTION="url" TARGET="frame" METHOD="método">*

## NOMBRES RESERVADOS

Aunque a primera vista no se presente la necesidad, en cuanto uno lleva algo de tiempo trabajando con *frames* se da cuenta que se necesitan desarrollar ciertas acciones para cada una de las cuales se ha definido un nombre reservado de ventana. Todos estos nombres reservados comienzan por el carácter de subrayado (_):

*_top*

Uno de los primeros inconvenientes del uso de *frames* sería que al pulsar sobre un enlace sin atributo *TARGET*, el nuevo documento aparece en la misma ventana o *frame* en que se encontraba el enlace reescribiéndolo (funcionamiento usual). De esta forma, si un usuario entra navegando en un servidor con *frames*, al salir del mismo continuará con los *frames* en su página y, aunque en el *frame* de navegación principal se encuentre en otro servidor, aún contará con los otros *frames* del primer servidor, quizá índices o logos. El problema se agrava al entrar en otro servidor con *frames*, lo que produciría un encadenamiento de los mismos, que aparecerían subdividiendo no toda la ventana sino el *frame* de navegación (en las figuras 2 y 3 se puede observar la realización de este efecto, en las que el documento *frame* que se ha desarrollado como ejemplo en apartados anteriores divide ahora el *frame* de navegación en lugar de la ventena completa del browser). Así sucesivamente, se iría acumulando información de distintos servidores en pantalla y haciendo cada vez más pequeña la zona útil de navegación. Para evitar este problema, es para lo que se define el nombre reservado para *TARGET*, *_top*, que implica que el URL apuntado por el enlace ocupará el cuerpo completo de la ventana del browser destruyendo la actual estructura de *frames* si la hubiere. Este nombre debe ser empleado en todos los enlaces que el lector coloque en su servidor apuntando hacia otros servidores. Por ejemplo:

**El efecto de celdas debe ser logrado mediante concatenación de framesets**

*<A HREF="http://highland.dit.upm.es:8000" TARGET="_top">Echeva's Server</A>*

*_self*

Es un nombre que obliga a que el documento señalado por el enlace aparezca en la misma ventana en la que se encuentra el enlace. Esto a primera vista parece algo inútil, ya que éste es el comportamiento habitual. Sin embargo, deja de serlo cuando se emplea una etiqueta *<BASE TARGET=>*. Por ejemplo, en el directorio de servidores que el autor tiene en su servidor bajo el nombre de "Paseos por Internet", todas las páginas índice disponen de una etiqueta *<BASE HREF=_top>* ya que los enlaces apuntan a otros servidores. Sin embargo, a la hora de definir el enlace correspondiente a la página del autor surge un problema: si no se indica *TARGET*, la página aparecerá ocupando todo el cuerpo de la ventana según define la etiqueta *BASE*, lo que romperá la estructura de *frames* del servidor sin haber salido de éste. Si por el contrario se indica *TARGET=main* (pues *main* es el nombre que el autor ha puesto a la ventana principal), surgirá otro problema: si el cliente no ha entrado por la página principal del servidor (por ejemplo, por disponer de un *bookmark* al directorio de servidores) o si ha optado por la opción de no emplear *frames*, no dispondrá en su ventana de un *frame* con el nombre de *main*, con lo que se producira el, esta vez indeseable, comportamiento de la generación automática de una nueva ventana. El destino *_self* viene a solucionar este problema: si el enlace estaba en un *frame* para ese usuario, el destino aparecerá en el mismo *frame*; si el enlace estaba en una página que ocupaba toda la ventana, el destino hará lo mismo.

*_blank*

Simplemente provoca que el documento apuntado aparezca en una nueva ventana en blanco sin nombre.

*_parent*

Provoca que el documento apuntado aparezca en el *FRAMESET* inmediatamente anterior en la jerarquía de *frames*. En caso de que no exista, actúa igual que *_self*

## ¿RECURSIÓN INFINITA?

En puntos anteriores se ha explicado la posibilidad de encadenar conjuntos de *frames*, ya sea a través de un documento *frame* con *FRAMESET*s encadenados o mediante una navegación en la que dentro de un *frame* aparecía un nuevo documento de tipo *frame*. Existe otra forma adicional de realizar este encadenamiento que consiste en facilitar como URL de un *frame* otro documento de tipo *frame*. Esta característica podría ser un problema si un desarrollador de páginas WWW malicioso decidiera facilitar como URL de un *frame* el correspondiente a su padre, por ejemplo, sea el documento *travieso.html* que contiene:

```
<FRAMESET rows="50%,50%>
<FRAME SRC="travieso.html">
<FRAME>
</FRAMESET>
```

Este documento provocaría una recursión infinita que podría acabar con la memoria del cliente. Por ello, un browser que implemente *frames* correctamente deberá prevenir casos como éste almacenando una lista con la jerarquía de *frames* para no permitir la llamada a un padre desde un hijo. En el caso de Netscape, cuando esto se produce, se muestra un *frame* en blanco, como si careciera de URL.

## CONTROL DE FRAMES EN NETSCAPE NAVIGATOR 2.x y NETSCAPE ATLAS

Pese a que el uso de los *frames* es una idea de Netscape y fue Navigator 2.x el primero en implementarla, la interfaz de usuario presenta ciertos problemas en esta versión. Así, el URL que se toma como actual es el correspondiente al primer documento *frame* que se ha encontrado, con lo cual, aunque se esté navegando desde hace tiempo por otros servidores, siempre que se conserve la estructura de *frames* de la página, una pulsación del botón "back", devolverá al documento que se visitó anteriormente al documento *frame* y no a aquel que aparece en el *frame* en el que se está navegando (este efecto se puede también observar en las ventanas de *Location* y título de las ilustraciones). Netscape previó esto, ya que mediante la pulsación del botón derecho del ratón sobre un *frame* aparece un menú que contiene, entre otras, la opción "back in *frame*". Sin embargo, este método ha demostrado no ser "amigable", por lo que ha variado en la última versión del browser de Netscape, en fase beta a la hora de escribir este artículo: Netscape Atlas, en la que sí se puede hacer uso del botón "back".

Asimismo, en las versiones 2.x, si se desea ver el código HTML de un documento "document source", se mostrará el del documento *frame*, y no el correspondiente al del documento que aparece en el *frame* de navegación, lo que presenta el problema añadido de que, aunque se tenga activada la opción "show location", el URL que se presenta es el correspondiente al documento *frame*, como se citaba en el párrafo anterior. Este problema también ha sido resuelto en Netscape Atlas.

**LISTADO 1**

```
<HTML>

<HEAD>
<TITLE>ECHEVA'S WEBTALK</TITLE>
</HEAD>

<FRAMESET ROWS="80,150,75,*">

<NOFRAMES>

<h1>IMPORTANT:</h1>

<TABLE BORDER>
<TR>
<TD>
TABLA: Etiquetas correspondientes al uso de frames
<FRAMESET [ROWS=lista|COLS=lista]>
<FRAME SRC= NAME= MARGINWIDTH= MARGINHEIGHT= SCROLLING= NORESIZE=>
<NOFRAMES>

<B>WebTalk</B> is designed to run on Netscape 2.0 or better.<p>
A Netscape 1.1N browser or better is <b>required</B> since it makes use of
<I>server-push</I>.<p>
To achieve full performance Netscape 2.0 is suggested.<p>
It uses <i>frames</i> feature from Netscape 2.0 to divide the screen in two
areas, one to write and the other one to read. If you're using an older
version of Netscape you should open two windows. Then, get this page
with both of them and press <a href=/cgi-bin/nph-etalk?10036>here</a>
in the first one and <a href=/talk/talkform.html>here</A> in the second one.
</TD>
</TR>
</TABLE>

</NOFRAMES>

 <FRAME NAME=title SRC=/talk/title.html>
 <FRAMESET COLS="115,*">
                    <FRAME NAME=video SRC=http://highland.dit.upm.es:8000/cgi-bin/nph-showvideo?10040>
                    <FRAME NAME=dialog SRC=/cgi-bin/nph-etalk?10036>
 </FRAMESET>
  <FRAME NAME=commands SRC=/talk/talkform.html>
 <FRAME>
</FRAMESET>

</HTML>
```

**Ejemplo de uso de concatenación de conjuntos de frames, frames con nombre y etiqueta <NOFRAMES>.**

## CONCLUSIÓN

Con el presente artículo se ha vuelto a saltar a otra de las ramas de las tecnologías WWW, el HTML. En este punto aún quedan otros temas por tratar, como el uso de los *client-side imagemaps*, pero también queda mucho aún que decir sobre otros temas como animaciones o documentos dinámicos en el lado del servidor (*server-push*), lo que conduce de nuevo al tema de desarrollo de CGIs, introduciendo los *NPH* (*No Parse of Headers*), sin olvidar ni dejar de lado el *VRML*, *Moving Worlds* o *Java*.

## CONTACTAR CON EL AUTOR

Para cualquier duda, comentario, sugerencia o crítica, se anima al lector a que se ponga en contacto con el autor mediante:

E-mail internet: echeva@dit.upm.es
E-mail Compuserve: 100646,2456
http://highland.dit.upm.es:8000

Dado el número creciente de mensajes recibidos se ruega al lector paciencia a la espera de la respuesta y si ésta no se produce que reenvíe su mensaje.

## NOTA DEL AUTOR

El talk para web que se muestra en el artículo se encuentra en versión alfa, por lo que el usuario no tiene conocimiento de entradas y salidas al foro ni de las personas conectadas. Por el momento (y mientras no tenga más tiempo) se trata únicamente de un juguete, aunque funciona.

# UNIDADES DE COMPILACIÓN (I)

*José C. Remiro*

Este artículo, junto con él del próximo mes, pretende implementar un conjunto de procedimientos y funciones para el tratamiento de expresiones aritméticas. En el presente artículo se describirán cada una de las estructuras que servirán para almacenar y manipular expresiones. Además, éstos servirán para presentar operaciones básicas sobre dos tipos de datos muy comunes en programación: las listas lineales y los árboles binarios.

## Para usar procedimientos y funciones de otros programas, Turbo Pascal proporciona la posibilidad de crear Unidades

Cuando un programador se enfrenta a una aplicación de cierta envergadura, es necesario que divida ésta en varias partes lógicas. Una de las características más importantes para decidir que un programa está bien diseñado consiste en que esté formado por módulos relativamente independientes que se puedan compilar de forma separada, pudiéndose de esta forma comprobar el correcto funcionamiento de cada módulo por separado, para posteriormente enlazar cada uno de estos módulos y proceder a la construcción del programa final.

El anterior método de trabajo se ve facilitado por la compilación independiente, ésta técnica se encuentra vinculada al compilador utilizado, siendo diferente para cada uno de ellos. A continuación, se introducirá una breve descripción de como funciona esta técnica en Turbo Pascal, el compilador que sirve de base a los programas de ejemplo.

Una primera posibilidad de utilizar un conjunto de procedimientos y funciones que ya han sido implementados para otros programas consiste en servirse de un editor, copiar y pegar el texto de los procedimientos y funciones que interesan para el programa que se está desarrollando. Esta estrategia no resulta muy adecuada, pues tales procedimientos pueden utilizar tipos, constantes y variables declaradas en el programa original que tendrían que ser descritos en el programa a construir. Para resolver estos inconvenientes, Turbo Pascal proporciona la posibilidad de crear unidades.

Una unidad se compone básicamente de cuatro partes (la estructura de una unidad se muestra en el cuadro 1), a saber:

- La declaración de tipos, constantes y variables que utilizarán los diferentes procedimientos y funciones de la unidad.
- La descripción de la interface de la unidad, que es una lista de los

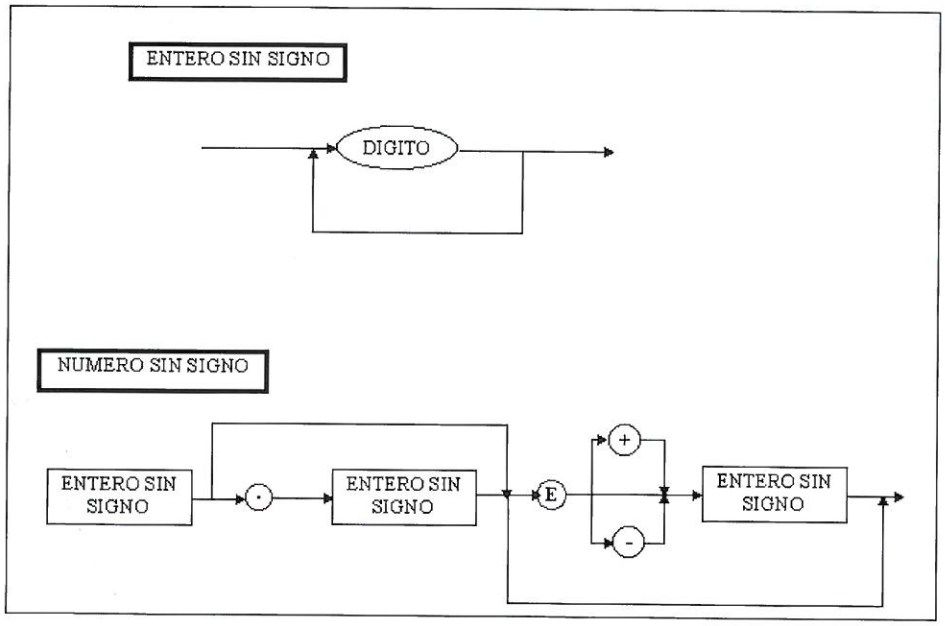
**La reutilización del código junto con la compilación separada hacen de las unidades de compilación un elemento importante dentro de la programación. Este mes y el próximo se implementarán funciones para el tratamiento de expresiones aritméticas.**

```
UNIT nombre_de_unidad;

INTERFACE
    · declaraciones (constantes, tipos, variables...) visibles
      desde los programas que utilicen la unidad

    · lista de cabeceras de funciones y procedimientos
      utilizables por los programas que utilicen la unidad

IMPLEMENTATION
    · declaraciones (constantes, tipos, variables...) no visibles
      desde los programas que utilicen la unidad. Apoyan, la
      implementación de los procedimientos y funciones de la
      unidad.

    · funciones y procedimientos junto con sus instrucciones

BEGIN
    · lista de instrucciones para la inicialización de la unidad

END.
```

**Cuadro 1.**

nombres de procedimientos y funciones, junto con sus respectivos parámetros, que podrán ser utilizadas por cualquier programa que indique la intención de utilizar la unidad mediante la cláusula *uses*.

- La implementación, donde se describen los procedimientos y funciones del interface. Esta parte de la unidad puede incluir procedimientos que no aparezcan en el interface, quedando ocultos al programa que utilice la unidad.
- La inicialización consiste en un conjunto de instrucciones donde se asignan valores iniciales a las variables de la unidad.




**Cuadro 2.**

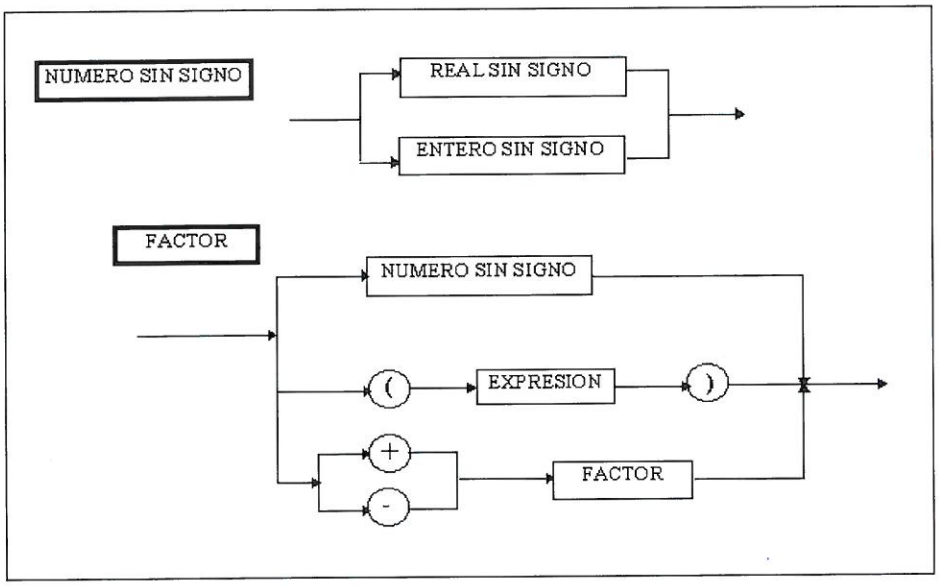
Una vez que se ha creado una unidad, ésta se compilará independientemente del programa que la utilice. Cuando un programa desee utilizar los procedimientos y funciones del interface de la unidad bastará con que lo indique al inicio del programa mediante la cláusula *uses* y el nombre de la unidad. Pero, además, existe una ventaja al utilizar las unidades: si un programa utiliza una unidad y se desea modificar la implementación, seguirá siendo válido el programa mientras no se vea modificado el interface. Para reflejar los cambios bastará con compilar de nuevo la unidad y el programa.

## EXPRESIONES ARITMÉTICAS

La naturaleza de la información que se va a tratar en el programa es una secuencia de operadores aritméticos y números sintácticamente correcta. Para definir la corrección de una expresión aritmética se han utilizado una serie de grafos sintácticos que se muestran en los cuadros 2, 3 y 4. Una expresión será correcta si se puede generar a partir del conjunto de grafos que definen su sintaxis.

Por ejemplo, el grafo que se asocia a la definición de término puede leerse de la siguiente forma: un término se compone de al menos un factor. Si hay más factores, entonces entre cada dos de ellos se encontrará o bien un asterisco o bien una barra inclinada.

El tipo de grafo sintáctico utilizado, está compuesto de dos tipos de elementos: por un lado los elementos terminales (rodeados por una circunferencia en los grafos), estos elementos no se refieren a ningún otro grafo, es decir, se interpretan tal cual. Así, un dígito es un elemento del conjunto {0,1,2,3,4,5,6,7,8,9}. Por otro lado, se encuentran los símbolos no terminales, que se definen a partir de otros. Así, una expresión simple no es un símbolo terminal, pues es necesario para definirlo recurrir al grafo sintáctico correspondiente a un término.




**Cuadro 3.**

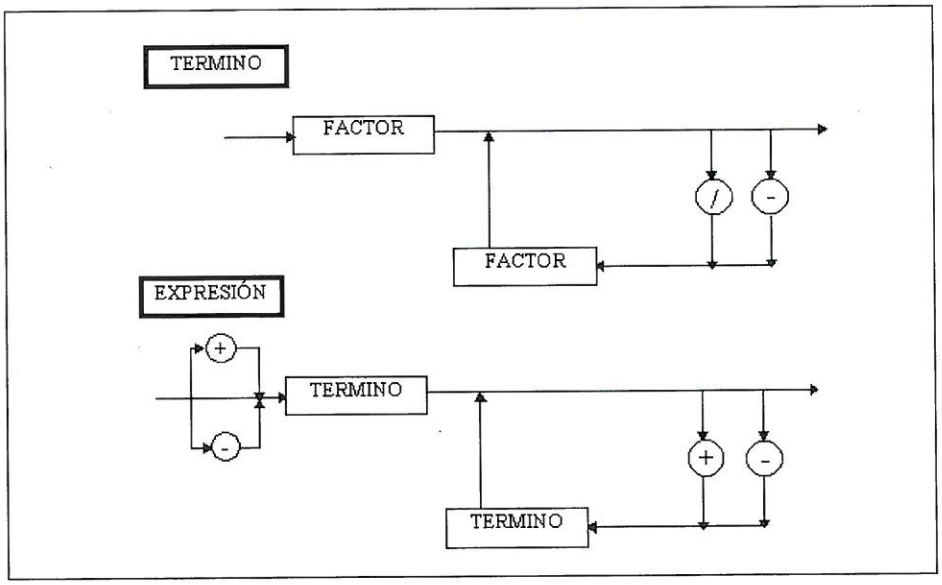
Además, la estructura de los grafos que definen la corrección de una expresión aritmética no es gratuita, pues para reconocer una expresión de longitud a priori indeterminada es necesario saber en cada momento del análisis sintáctico a qué grafo hay que dirigirse para reconocer a cada uno de los elementos de la expresión. Para cumplir este requisito, los grafos han sido diseñados de tal forma que conociendo un elemento actual y el siguiente se pueda establecer la categoría a la que pertenecen. Esto implica que una vez que se ha decidido que un elemento de la expresión se corresponde con un elemento del grafo no habrá marcha atrás.

Por ejemplo, la expresión 2*(3 + 5) aplicada al grafo correspondiente a expresión, se haría corresponder en su totalidad como un término, puesto que el elemento que sigue al 2 no es ni el símbolo terminal + ni el símbolo terminal - (véase cuadro 3). Siguiendo con el análisis, en el grafo correspondiente a término, el 2 se asociaría a un factor y (2 + 3) estaría asociado a otro factor, pues el elemento que sigue a 2 es un asterisco. Posteriormente, se procedería a comprobar que 2 y (3 + 5) son factores. Pero 2 se corresponde con la descripción de un número sin signo. Posteriormente se pasaría a comprobar si se trata de un entero sin signo o de un real sin signo y así sucesivamente.

Para el factor (3 + 5), se reconocería por el paréntesis abierto que su interior debería ser una expresión, por lo que se pasaría de nuevo a comprobarlo a través del grafo dedicado a definir expresiones.

Todo este mecanismo no seria posible si cada vez que se hace corresponder una expresión al grafo correspondiente no se pudiera decidir entre las diferentes opciones que presenta el grafo. Así, para reconocer (3 + 5) mediante el grafo factor no hay más posibilidad que 3 + 5 sea una expresión, pues el elemento inicial de la expresión es un paréntesis abierto.

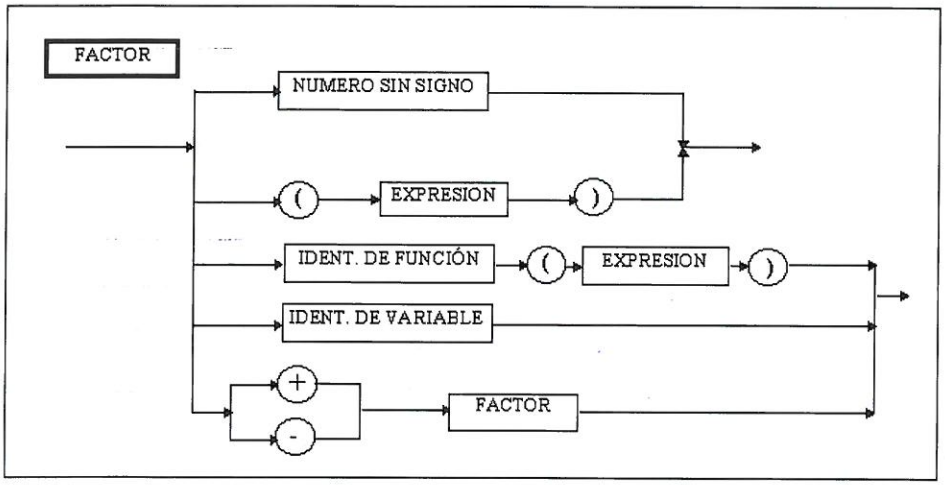
## Una forma natural de representar una expresión es una lista lineal




**Cuadro 4.**

Si, por ejemplo, se deseara introducir en el conjunto de expresiones la utilización de funciones matemáticas como senos, logaritmos etc., se debería modificar el grafo factor como se indica en el cuadro 4. No habrá problemas mientras el identificador de función comience por un carácter distinto de un número o de un signo (para diferenciarlo de un número sin signo o de los signos + y - ). Por otro lado, si se desea introducir en las expresiones identificadores de variable (secuencia de caracteres que comienzan por un carácter alfabético), éstas serán diferenciadas de los identificadores de función gracias a los paréntesis obligatorios que deben acompañar al identificador de función.

Como se describirá posteriormente, los grafos se corresponderán con procedimientos de comprobación de una parte de la expresión. La aparición de varios elementos no terminales iguales en diferentes




**Cuadro 5.**

grafos se corresponderá con llamadas mutuamente recursivas entre procedimientos. Los símbolos no terminales servirán para establecer a qué procedimiento hay que llamar en cada momento del análisis.

## ELEMENTO A DE UNA EXPRESIÓN

Como se comentó con anterioridad, una expresión aritmética era una secuencia de longitud indeterminada en la que se combinaban tanto números reales como operadores aritméticos. Este hecho implica que una forma natural de representar una expresión es una lista lineal cuyos nodos incluyan bien un número, o bien un operador aritmético (representados por los caracteres +, -, *, /). Dado que no existe ningún tipo predefinido por el lenguaje que pueda almacenar esta información habrá que crear un nuevo tipo, que se denominará *nodo*.

El tipo *nodo* es un registro de tipo variable con un campo de selección denominado *tipo_nodo* perteneciente a un tipo enumerado *clase_nodo*. Dependiendo del valor de dicho campo de selección se puede conocer el tipo de elemento que almacena un nodo en particular. De esta forma, si *tipo_nodo* contiene el valor *número*, el nodo contendrá un número real, si por el contrario contiene el valor *operación*, el nodo contendrá un carácter que representará al operador aritmético que almacenará el nodo. En el cuadro 6 se muestra la definición de un nodo y sus operaciones asociadas.

Parece buena idea construir un conjunto de procedimientos y funciones para representar las operaciones mínimas que se pueden realizar con un nodo y todo el conjunto incluirlo en una unidad, de tal forma que cualquier programa que utilice la unidad *u_nodo* pueda utilizar el tipo de datos a través de dichas operaciones. Las operaciones básicas que se ha decidido implementar en esta unidad, son las siguientes:

- *Es_Numero.*- función que admite como parámetro un nodo, devuelve el valor *true* cuando el nodo contiene un número y devuelve el valor *falso* en otro caso.
- *Almacena_operacion.*- Procedimiento que admite un nodo y un carácter donde se supone que hay un operador. Almacena en el nodo la operación.
- *Almacena_numero.*- Procedimiento de similares características que el anterior, pero que almacena un número real.
- *Devuelve_operacion.*- Función que admite un nodo que contiene supuestamente una operación, devuelve la operación almacenada en el nodo.
- *Devuelve_numero.*- Función de similares características que la anterior, pero para un nodo que contiene un número.
- *Visualiza_nodo.*- Procedimiento que visualiza en pantalla el contenido de un nodo.

**Una expresión será correcta si se puede generar a partir del conjunto de grafos que definen su sintaxis**

## IMPLEMENTACIÓN DE EXPRESIONES

Una vez construida la unidad que implementa los nodos y sus operaciones básicas, se puede construir la unidad que representa a las listas lineales que almacenarán la totalidad de una expresión.

```
clase_nodo = (numero, operacion);
nodo = RECORD
     CASE tipo_nodo : clase_nodo OF
          numero  :  (num: real);
          operacion : ( oprcion: char);
     END;


FUNCTION Es_numero( nodo_param: nodo): Boolean;

PROCEDURE Almacena_operacion ( VAR nodo_param: nodo; cont_oprcion: char);

PROCEDURE Almacena_numero  ( VAR nodo_param: nodo ; cont_num: real);

FUNCTION  Devuelve_operacion ( nodo_param: nodo ): char;

FUNCTION  Devuelve_numero  ( nodo_param: nodo ): real;

PROCEDURE Visualiza_nodo  ( nodo_param: nodo);
```

**Cuadro 6.**

El tipo lista (véase cuadro 7) no es más que un puntero capaz de almacenar la dirección donde se encuentra un nodo de la lista, que a su vez se compone de dos informaciones: un puntero al siguiente nodo de la lista y el contenido del nodo actual de tipo *nodo* (es decir, puede contener o bien una operación o bien un número).

Las operaciones básicas que se pueden esperar para la utilización de las listas son las siguientes:

- Crea_nodo_lista.- Esta función devuelve un puntero que apunta a un nodo que contiene el valor que se le ha pasado como parámetro. Se utilizará para generar cada uno de los nodos de la lista.
- Lista_vacia.- Esta función devuelve el valor *true* si la lista está vacía y *false* en cualquier otro caso.
- Libera_nodo_lista.- Esta función devuelve la memoria ocupada por un nodo que ya no se utiliza.
- Ins_inicio_lista.- Esta función permite insertar un nuevo nodo al inicio de la lista.
- Siguiente_nodo.- Esta función, dado un puntero, devuelve el nodo al que apunta el puntero.
- Long_lista.- Esta función, dada una lista, devuelve la longitud de la misma.
- Invierte_lista.- Este procedimiento invierte los nodos de la lista pasada como parámetro.
- Visualiza_lista.- Presenta en pantalla el contenido de la lista pasada como parámetro.

## ÁRBOLES Y EXPRESIONES

Hasta ahora la implementación de las unidades para almacenar y manipular expresiones no ha presentado ninguna dificultad, pues una lista es una estructura natural para representarlas.

El objetivo principal que se persigue, sin embargo, es el de evaluar una expresión y parece bastante complicado realizar esta operación si ésta se representa como una lista, pues el proceso de evaluación no es lineal. La prioridad de las operaciones y los paréntesis afectan a la evaluación de una expresión.

Así, si se evalúa la expresión 4+2*(13+5)/(7-1), primero habrá que evaluar las subexpresiones encerradas entre paréntesis, quedando la expresión como 4+2*18/6, a continuación se realizarán los productos y las divisiones, quedando la subexpresión intermedia 4+6 y por último las sumas y restas, resultando la evaluación de la expresión el valor 10.

Por tanto, parece ser que una lista y sus operaciones asociadas no son suficientes para poder evaluar una expresión, o bien no sería la estructura más aconsejable a utilizar. Una de las soluciones a este problema consiste en disponer la expresión en un árbol, de tal forma que cada nodo que posea hijo izquierdo e hijo derecho distintos de *nil* (las operaciones aritméticas que se utilizan son todas binarias), contenga una operación, mientras que las hojas de los árboles contengan los números a los que se aplicarán las operaciones. El cuadro 8 muestra la representación de la anterior expresión mediante un árbol.

Es evidente que si se puede realizar la construcción de un árbol de las anteriores características a partir de una lista que contenga una expresión sintácticamente correcta, el proceso de evaluación de la expresión se vería reducido considerablemente, pues bastaría con ir evaluando las operaciones de los niveles más profundos e ir ascendiendo hasta llegar a la raíz (las operaciones que tienen mayor prioridad ocupan niveles inferiores en el árbol).

Los diferentes procedimientos y funciones que se proponen para el interface de la unidad *u_arbol* son los siguientes:

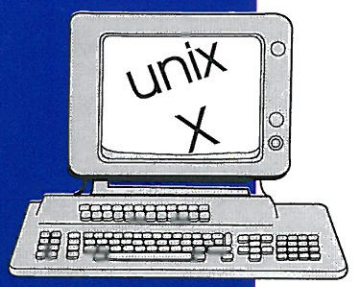
- Crea_nodo_arbol.- Función que crea un nodo cuyo contenido es el parámetro asociado.
- Libera_nodo_arbol.- Procedimiento para liberar la memoria utilizada por un nodo del árbol.
- Anyade_hijo-izdo.- Dispone uno de los árboles pasado como parámetro al procedimiento como hijo izquierdo del otro árbol.
- Anyade_hijo_dcho.- Similar al anterior pero como hijo derecho.
- Obten_hijo_izdo.- Esta función devuelve el hijo izquierdo del árbol pasado como parámetro.
- Obten_hijo_dcho.- Similar a la anterior pero con el hijo derecho.
- Visualiza_arbol.- Visualiza en pantalla los elementos del árbol.

## CÓMO COMPILAR UNA UNIDAD

El proceso de compilación de una unidad es similar al de un programa, pero el archivo generado tendrá la extensión *.TPU*.

Para que el compilador pueda encontrar las unidades que utilice un determinado programa y puesto que no es posible indicar una trayectoria en la cláusula *uses*, donde se encuentran las unidades, o bien éstas se sitúan en el mismo directorio que el programa fuente, o bien se modifica dentro de la opción del compilador *Options->Directories* el directorio donde se sitúan las unidades.

# PLAN 9: ¿EL SUCESOR DE UNIX?




**¿Qué puede ocurrir cuando se reune un grupo de gente entre los que se encuentran Ken Thompson, Dennis Ritchie y Robert Pike con el firme propósito de trabajar? ¿Qué puede ocurrir si, además, lo hacen bajo el techo de AT&T Bell Labs? Probablemente que resulte un nuevo sistema operativo que presente una red de máquinas diversas como un único macrosistema. En AT&T Bell Labs, se formularon estas dos preguntas. El resultado: Plan 9**

*Fernando J. Echevarrieta*

Tres o cuatro años atrás, algo empezó a gestarse en el corazón de los Laboratorios Bell, el núcleo de investigación y desarrollo de AT&T, de donde, como otras muchas tecnologías, surgió UNIX de la mano de Ken Thompson y Dennis Ritchie. Y fue precisamente un grupo de desarrolladores, entre los que se encontraban los propios creadores de UNIX, el que comenzó a trabajar en un nuevo proyecto de Sistema Operativo Distribuido denominado *Plan 9* en honor a una de las peores películas de ciencia ficción de la historia: *Plan 9 from Outer Space* de Ed Wood.

Hoy en día, Plan 9 es una realidad ofrecida a usuarios individuales e instituciones académicas para uso no comercial por el *Centro para el Desarrollo Informático de Bell Labs*, antiguo *AT&T Bell Labs*, que pertenece a *Lucent Technologies*, nombre con el que desde el primer trimestre del presente año se conoce a *AT&T Network Systems*.

Plan 9 constituye un sistema operativo, escalable, eficiente en el uso de recursos y con un uso seguro de la red. Se trata de un sistema operativo "distribuido" a efectos "realistas". Es decir, no llega a proporcionar la transparencia total que exigía Tanembaum para calificar como tal a un S.O.D. (Sistema Operativo Distribuido), pero sí se trata de un sistema en que, al igual que existen servidores de ficheros, existen servidores de CPUs y en el que el acceso y tratamiento de recursos locales y remotos se realiza de la misma forma tanto interna como externamente.

## LA HERENCIA DE UNIX

La idea original consistía en realizar un moderno sistema operativo partiendo desde cero, no como un repuesto o actualización de UNIX. Debería realizarse un S.O.D. que respondiera a las necesidades y entorno tecnológico actuales. Y aunque así ha sido, naciendo donde ha nacido y vieniendo de la mano de los creadores de UNIX, habría sido ilógico que se arrojara a la papelera toda la experiencia y las buenas ideas incluidas en este S.O. Sin embargo, no se debe olvidar que UNIX nació hace más de un cuarto de siglo y ha llegado a ser lo que es tras una continuada evolución. En su origen, no se trataba de un sistema operativo de red, no incorporaba la seguridad como una de sus directivas principales y no se trataba de un sistema orientado a gráficos. Por otra parte, si bien se le han ido añadiendo estas funcionalidades, no constaban en su diseño inicial, realizado con una metodología diferente de la actual, sobre todo, en el diseño de los S.O.D. Aún así UNIX incorpora tantas buenas ideas que hoy en día más del 80% de los ordenadores de internet emplean este S.O.

De este modo, la cultura de la empresa respecto a UNIX (y, por supuesto, toda la experiencia acumulada aparte de UNIX) sirvió para no olvidar ninguna de sus ventajas y para, desde el diseño incial de Plan 9, suplir todas sus carencias. Era necesario desarrollar un S.O.D. concebido para ser escalable y ejecutarse en entornos de red dinámicos, para lo que debería ser fácilmente adaptable. Por otra parte, la plena potencia de Plan 9 como S.O.D. se demuestra (obviamente) cuando se instala sobre una red y no sobre una única máquina, ya que se adapta para asignar funciones especia-

lizadas a cada elemento de la misma.

Entre las características que comparte con UNIX, se puede citar:

- Se encuentra escrito en C, lenguaje que ha demostrado ser adecuado para la programación de este tipo de sistemas, con lo que se facilita su portabilidad.
- La interfaz de usuario es sencilla y potente, ofreciendo un refinamiento de la *shell* de UNIX aunque, como en este caso, su sencillez y potencia no implican una especial facilidad de uso. Sin embargo, no se trata de un S.O. pensado para colocar en los ordenadores personales de todos los hogares sino, como se ha mencionado, en redes.
- Dispone de un sistema de ficheros jerárquico en el que todo se encuentra anclado en la raíz. Un sistema de ficheros, en sus capas inferiores, conceptualmente casi idéntico al de UNIX. En él también se montan y desmontan volúmenes, incluso con la misma orden *mount*.
- Para Plan 9, como para UNIX, el universo se encuentra constituído por ficheros. Como en el segundo caso, todo son ficheros, incluso los dispositivos, que se gestionan mediante operaciones de entrada y salida. Pero Plan 9, como se verá más adelante, potencia esta idea hasta el límite.

## IDEAS PROPIAS

Plan 9 exige la existencia de un terminal con un sistema de ventanas que, de forma jocosa, han denominado "8 1/2" (ocho y medio). Este terminal es un elemento lógico que puede ser cualquier ordenador de la red funcionando con Plan 9 con una CPU que ejecute programas y con un sistema de ficheros "propio", pero se caracteriza por servir como interfaz de acceso al sistema para un usuario humano.

El procedimiento de instalación del sistema analiza la red y hace de la interconexión entre máquinas una ventaja, asignando de forma óptima funcionalidades específicas a cada nodo.

Dispone de un sistema de ficheros con "vistas de usuario". Es decir, como todo sistema operativo moderno, se trata de un sistema multiusuario, pero no se limita como UNIX a mostrar un único sistema de ficheros a todos los usuarios distinguiendo su forma de acceso en base a un sistema de permisos, sino que, en Plan 9, un mismo sistema de ficheros puede ser visto de forma completamente distinta por cada usuario del sistema, ya que define una capa superior de "vistas" para el mismo. Esta capa superior recibe el nombre de "*Espacio de nombres*".

La gestión interna de todos los recursos es igual, con lo que desaparece para siempre la diferencia entre recursos locales y remotos. Para ello incorpora en su núcleo un protocolo de acceso a recursos denominado *P9*.

## REQUERIMIENTOS

Por el momento, Plan 9 ejecuta sobre plataformas Intel 386 o superior, MIPS, SPARC y Motorola 68020. Salvo ciertas partes del kernel dependientes de la arquitectura, los compiladores y algunas librerías, el resto de las fuentes es común, lo que facilita su portabilidad a otros sistemas. Por el momento ya se estan realizando, por parte de terceros, ports para SGI Indy R4000, Sun 3/50 y 3/60 y DEC Alpha.

En el entorno PC, el kernel, el sistema de ventanas y las aplicaciones básicas se acomodan en una máquina con 8Mb de RAM. El kernel, el sistema de ventanas, un editor y el soporte básico de internet a través de ethernet caben en un simple disquete. El sistema básico para desarrollo cabe en cuatro disquetes que se pueden conseguir gratuitamente para evaluación.

La versión completa con fuentes en CD-ROM requiere 500 Mb de disco y, aunque el CD-ROM contiene 375 Mb de ficheros, si sólo se emplean PCs no hay necesidad de instalar los directorios */mips*, */sparc* y */68020*, con lo que se ahorran 140 Mb. Del mismo modo, si no se necesitan las fuentes (*/sys/src*), se ahorran otros 43 Mb, con lo que se podría realizar una instalación "apretadita" en unos 200 Mb.

En el otro extremo, se pueden realizar instalaciones como la que emplea AT&T con varias máquinas multiprocesador SGI Power Series y Challenge como servidores de CPUs y un jukebox WORM de 350 Gb para el servidor de ficheros.

## EL ESPACIO DE NOMBRES

Como se ha mencionado anteriormente, cada proceso Plan 9 dispone de una vista particular del sistema de ficheros que se denomina "espacio de nombres". Así, dos procesos pueden diponer de dos vistas completamente diferentes del mismo sistema de ficheros. En el caso de UNIX todos los procesos disponen de la misma vista del sistema de ficheros, con la única diferencia de que pueden variar sus permisos de lectura, escritura o ejecución sobre el mismo. El único ejemplo similar que se puede encontrar en el mundo UNIX a una vista de Plan 9 es el caso del dispositivo */dev/tty*. Este dispositivo se encuentra "cableado" en el kernel de UNIX, por lo que cada proceso ve un */dev/tty* diferente que corresponde al terminal al cual se encuentra asociado. Plan 9 lleva este comportamiendo al extremo ampliándolo a todo el sistema de ficheros aunque, en la práctica, los

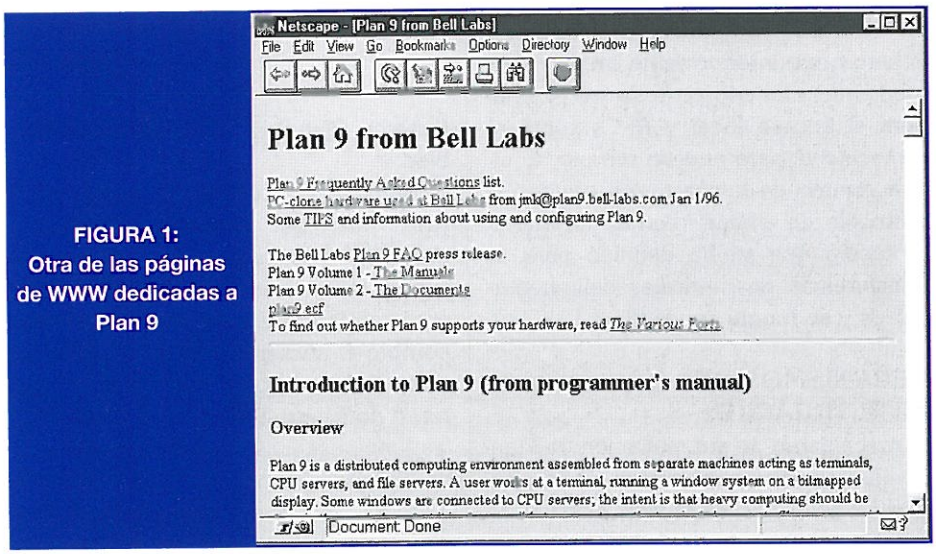



**FIGURA 1:**
**Otra de las páginas de WWW dedicadas a Plan 9**

espacios de nombres suelen ser compartidos por grupos de usuarios.

Todo espacio de nombres Plan 9 cuenta con una raíz / y bajo la misma se encuentra un directorio */bin* que surge para la vista de la unión de todos los directorios que contienen ejecutables para el proceso, generalmente los directorios de shell scripts y binarios del sistema y del usuario. De esta forma se elimina el uso de un PATH de búsqueda tan empleado en otros S.O. El orden de montaje de estos directorios para la definición de un orden de búsqueda es configurable para no perder funcionalidad.

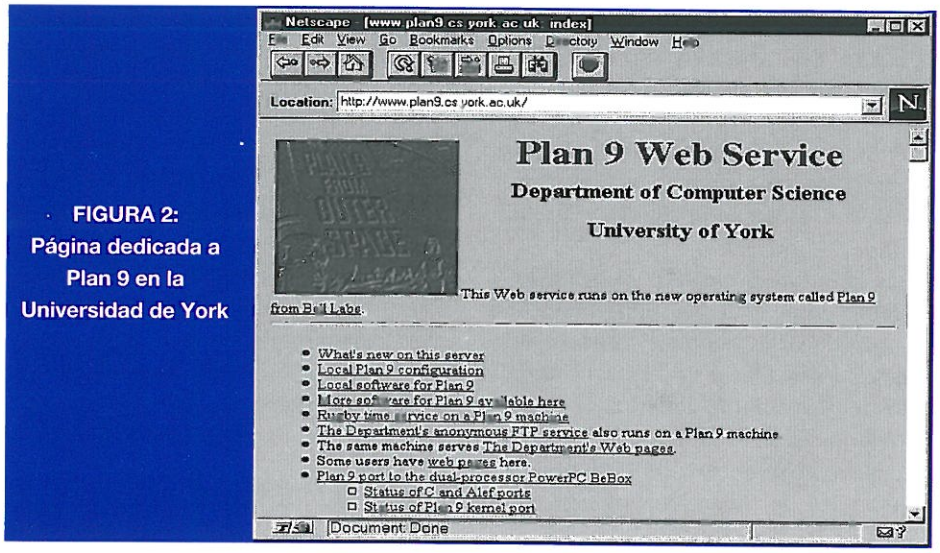



**FIGURA 2:** Página dedicada a Plan 9 en la Universidad de York

## PROTOCOLOS DE PLAN 9

El "sistema nervioso" de Plan 9 se encuentra constituido por el protocolo *9P*. Es este protocolo el que hace posible que el sistema pueda tratar todos los recursos como ficheros y que éstos se encuentren accesibles y se accedan por igual, ya sean locales o remotos. Es importante destacar el hecho de que "se accedan por igual". En sistemas basados en *NFS*, por ejemplo, sí se permite un acceso a ficheros remotos, pero *NFS* no se puede emplear para acceder a dispositivos ni para acceder a ficheros locales. Por este segundo motivo, UNIX dispone de dos métodos de acceso a recursos: uno a través de NFS para el acceso a ficheros remotos y otro a través de rutinas locales de su capa del kernel correspondiente al sistema de ficheros para acceso a ficheros locales y ficheros de dispositivos.

Plan 9 accede por igual a todos los recursos a través de 9P. El protocolo ofrece una semántica única de acceso a ficheros que interiormente implementa mediante procedimientos: del kernel para el acceso local y RPCs sobre el protocolo *IL* para acceso remoto. IL es el acrónimo de *Internet Link protocol* o protocolo de *enlace internet*, un nuevo protocolo que se ha definido para la transferencia de mensajes delimitados fiables y se monta sobre *IP*.

## MECANISMOS DE AUTENTICACIÓN

El mecanismo de autenticación de Plan 9 sigue el mismo esquema de diseño que *kerberos*. No se envían *passwords* por la red sino que, en su lugar, se emplea un mecanismo de tickets encriptados facilitados por un servidor de autenticación. En Plan 9 no existe el concepto de superusuario como en UNIX, ni tampoco la idea de programas con *setuid*.

El servidor de ficheros Plan 9 no corre programas de usuario, y sólo permite el acceso a ficheros protegidos desde la consola o a propietarios debidamente autenticados.

## SISTEMA DE FICHEROS PROC

Algunos UNIX como System V Release 4 y Linux, emplean un sistema de ficheros denominado *proc*. Mediante este tipo de sistema es posible acceder a procesos, entornos de proceso, dispositivos y recursos a través del sistema de ficheros. Plan 9 lo ha incorporado también.

Así, bajo el directorio */proc* de Plan 9 (que, por regla general, es el mismo para todos los procesos), existe otro subdirectorio correspondiente a cada proceso del sistema y que toma como nombre el número de proceso. Bajo él, otros ficheros como *ctl*, *mem*, *status*, *text* o *note* permiten acceder a través de las funciones E/S de la API del sistema de ficheros al control (*stop*, *start*...), memoria, información de status, fichero ejecutable, y lista de señales, respectivamente, correspondientes al proceso en cuestión. Así, por ejemplo, el comando ps se podría implementar en shell mediante

*cat /proc/*/status*

Por otra parte, los mensajes que se envían a través de lecturas o escrituras de ficheros son mensajes de texto. Con ello, si bien se puede perder algo en prestaciones, se gana en claridad y, además, se evita el problema de la traslación *host byte order* a *network byte order* y viceversa que nos preocupaba en anteriores artículos.

## ENTORNO Y CREACIÓN DE PROCESOS

Otra de las nuevas características de Plan 9 es la forma de acceder al entorno de un proceso que tambien se realiza a través del espacio de nombres a través del directorio */env*. Por ejemplo, un proceso con dos variables de entorno podría contener:

*/env/user*
*/env/pid*

cuyo contenido se podría ver con

*% cat /env/user*
*echeva*
*% cat /env/pid*
*456*

Nuevamente, el espacio de nombres será distinto para proceso (salvo excep-

**Para Plan 9, como para UNIX, todo el Universo se encuentra constituido por ficheros**

ciones entre hijos y padres). Así, una forma de ver todas las variables asociadas a un proceso podría ser:
*(cd env; echo *)*

Respecto a la creación de procesos, se realiza a través de un proceso padre a traves de la primitiva *rfork*. Esta llamada al sistema acepta un mapa de bits como argumento que define cómo se compartirán el espacio de nombres, el entorno y la tabla de descriptores entre padre e hijo.

## TRANSPARENCIA DE RED

Entre las características que otorgan a Plan 9 una gran funcionalidad se encuentran:

- *Transparencia y accesibilidad de sistemas de ficheros*

El servicio de transferencia de ficheros *ftp* no se enuentra disponible como un comando, sino como un sistema de ficheros. Así, por ejemplo, ejecutando:
*ftpfs -a echeva@dit.upm.es highland.dit.upm.es*
genera una conexión de *ftp* anónima a la máquina del autor enviando como *password* la dirección de *mail* del mismo. A partir de ese momento, se monta la jerarquía de ficheros de *ftp* anónimo de *highland* en el directorio */n/ftp* del espacio de nombres Plan 9 local. Así, para coger el programa cern-inst, facilitado en un número anterior, sería suficiente con teclear
*cp /n/ftp/pub/www/servers/cern-inst/cern-inst $home*
como si se tratara de un fichero "local"

Existe un servidor del protocolo 9P para UNIX con lo que, con colocar este servidor en las máquinas UNIX de una red, sus sistemas de ficheros se hacen inmediatamente accesibles a los sistemas Plan 9. Análogamente, existe un servidor NFS para Plan 9, con lo que los sistemas de ficheros de este sistema son también accesibles a otras máquinas con NFS, como entornos UNIX, DOS o Windows.

Asimismo, Plan 9 es capaz de leer sistemas de ficheros DOS/Windows, ISO 9660 CD-ROM y cintas *cpio*.

Es posible hacer coexistir en una máquina sistemas Plan 9, DOS, Windows y Windows 95.

- *Transparencia de CPUs*

Como se ha expuesto, uno de los elementos que adopta Plan 9 es la figura del servidor de CPUs. Al igual que se emplean servidores de ficheros o de autenticación, es posible realizar determinados trabajos empleando una CPU remota más potente en lugar de la del terminal. Para ello basta ejecutar un comando como:
*cpu -h nexus -c (cd $home/src ; mk)*

**El mecanismo de autenticación de Plan 9 sigue el mismo diseño que kerberos**

que ejecutaría en *nexus* una *shell* con el mismo espacio de nombres que el terminal que cambiaría al directorio del usuario en este espacio de nombres y realizaría una compilación mediante *mk*, el sustituto de *make*. De esta forma, se combinan las capacidades de *shell* remota que proporciona UNIX para ejecutar en otra CPU con las de NFS para que aquella CPU comparta el espacio de nombres. Sin embargo, en este caso, la transparencia no es total. Es el operador humano el que decide el uso de una u otra CPU y debe saber cuál quiere utilizar, no es decisión del S.O. Aquí es donde se aprecia un ejemplo de por qué no se trata de un S.O.D. puro de Tanembaum.

- *Transparencia de máquinas y redes*

Cada red se ve como un conjunto de ficheros sobre los que se pueden realizar operaciones de creación, entrada/salida y control. Para ello, la API de programación define un núcleo semántico común independiente de la naturaleza de la red subyacente y lo complementa con un servidor de traducción de direcciones de red a direcciones "legibles por el hombre". Con esta interfaz, los procesos de aplicación no tienen conciencia de si trabajan sobre una red TCP/IP, RDSI, sobre un modem o sobre cualquier otro tipo de red. Ni siquiera saben si la máquina sobre la que se ejecutan se encuentra en la red que están manejando, ya que los ficheros interfaz de red podrían haberse importado de otra máquina. Dicho de otra forma: como se vino indicando en los primeros artículos de esta sección, no hay que olvidar que la prin-

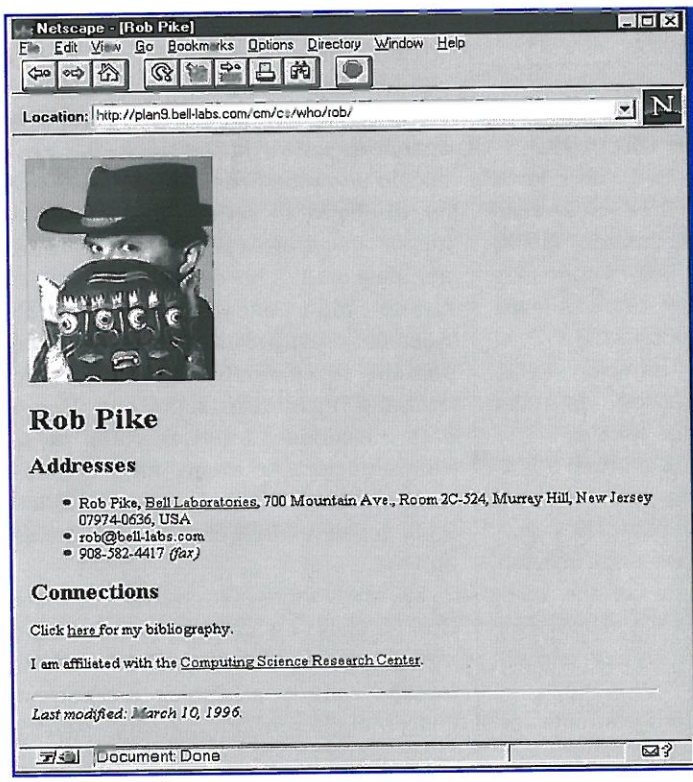



**FIGURA 3:** Robert Pike, una de las leyendas vivas del mundo de los S.O.

cipal función de un S.O. es definir una máquina virtual que oculta el hardware. En el caso de S.O.D la correspondencia máquina virtual-hardware no es uno a uno. Es decir, hay que pensar en la máquina virtual Plan 9 como un único elemento, aunque se encuentre formada por una red completa. Así pues, aúnque la máquina que contiene la CPU que está ejecutando el proceso que accede a una red no se encuentre conectada a esa red directamente, sí puede estarlo la máquina virtual Plan 9 a través de otro elemento de la red que oculta.

Por otra parte, todo proceso de entrada/salida puede ser "trazado" si se ejecuta sobre un servidor que observe este tipo de interacciones. Asimismo, el depurador es capaz de examinar un programa que se esté ejecutando en otra máquina aunque se trate de una arquitectura diferente.

## EL SISTEMA DE VENTANAS 8 1/2

El sistema de ventanas *8 1/2* destaca por su claridad y sencillez. Como idea fundamenteal proporciona un teclado, un ratón y una pantalla virtual a cada aplicación. De esta forma, aunque los procesos "creen" controlar estos periféricos directamente, son engañados por el sistema operativo, que es el único que accede a teclado, ratón y pantalla reales. Así, por ejemplo, todo el sistema puede correr recursivamente en una de sus ventanas. Una de las ventajas de esta perspectiva es que es posible ejecutar un servidor de 8 1/2 en una CPU de un servidor de CPUs y mostrar el sistema entero en una ventana del servidor local, ventana que se puede ampliar a toda la pantalla. De esta forma, todos los procesos se ejecutarán el en servidor de CPUs. Lo único que se le puede reprochar es la austeridad, aunque se traduce en una destacable eficiencia. El entorno de usuario Plan 9 se encuentra principalmente orientado a texto, no existen iconos ni *drag & drop* (arrastrar y soltar) en el sistema 8 1/2 aunque, por supuesto, se trate de un entorno gráfico y el sistema proporcione librerías gráficas para el desarrollo.

Otra de las características interesàntes es que permite retrasar la lectura que un proceso hace de la entrada aún cuando ésta conste de varias líneas. De esta forma, es posible corregir la entrada hasta que se considere correcta y sólo entonces dejar que el proceso la lea, lo que equivaldría a emplear un editor.

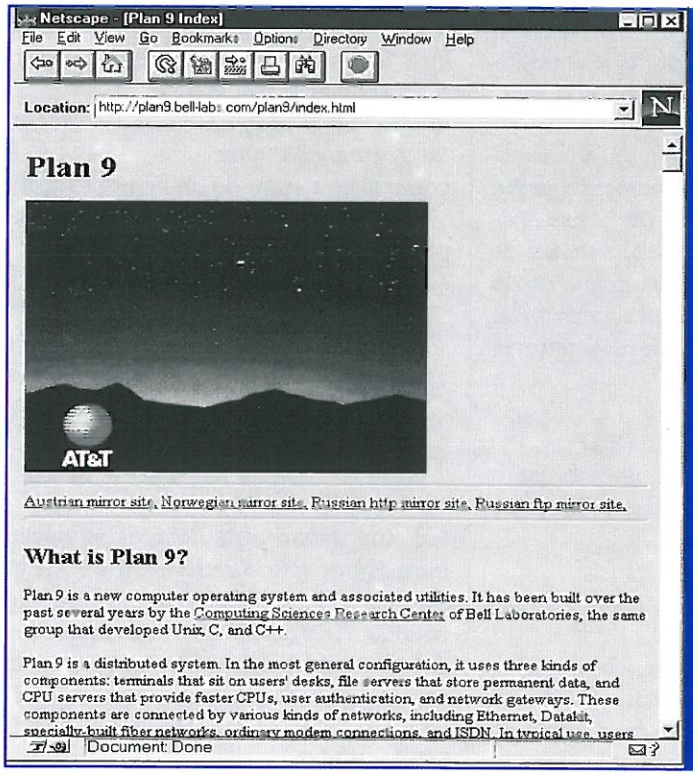



**FIGURA 4: Página home de Plan 9 en internet**

Aunque AT&T no tiene intención de trabajar sobre ello, terceros ya han realizado un *port* de X WINDOW para Plan 9 aunque, por el momento, sólo funciona sobre plataformas MIPS.

## APLICACIONES, ENTORNO DE DESARROLLO Y DE USUARIO

Entre las aplicaciones con que cuenta la distribución de Plan 9 se encuentran "las de siempre" del mundo UNIX, como *troff*, *awk*, *ghostscript* y todos los comandos de UNIX, y otras nuevas como el servidor de ventanas *8 1/2*, el editor/entorno de programación *acme*, el depurador *acid*, el sustituto de make *mk*, la shell *rc* y el editor *sam*.

Además, se incorpora un nuevo lenguaje de programación que recibe el nombre *Alef*. *Alef* admite *threads* y gestiona las comunicaciones entre máquinas y procesos mediante canales tipados y tipos abstractos de datos. Por supuesto, también se incluye un compilador de C.

*Acid*, el depurador programable, es capaz de seguir programas multiproceso y depurar programas presentes en otras máquinas y con otra arquitectura.

*Acme* proporciona una interfaz de usuario como editor y entorno de programación. Genera ventanas que se posicionan automáticamente con simples pulsaciones de ratón. Cada una de ellas dispone de una cabecera y un cuerpo. En la cabecera suele aparecer el nombre de un fichero y una serie de opciones que representan acciones y cambia dinámicamente en funcion del contexto. Por ejemplo, no aparece la opción de salvar hasta que se ha producido una modificación. En general se ha diseñado el sistema buscando la mayor transparencia posible, lo que, por otra parte, hace que al principio cueste algo acostumbrarse a él. Además, cada palabra que aparece en pantalla se convierte en un comando con sólo hacer click sobre ella. Como nota anecdótica, en el apartado de la programación, se mencionará que los códigos de retorno de los procesos Plan 9 pueden ser *strings* en lugar de simples enteros.

La shell *rc* es equivalente a una Bourne shell. La línea de comando es prácticamente idéntica a la de UNIX, aunque varía a la hora de programar. Las variables son, por defecto, *arrays* y se concatenan de forma diferente a la

Bourne shell. Como novedad, se puede acceder a la llamada *rfork* desde la shell.

La configuración del sistema se realiza mediante *shell scripts*, de la misma forma que en UNIX. Por ejemplo, la configuración de los servicios de los terminales se indica en el fichero */rc/bin/termrc*, y la de servidores de CPUs en */rc/bin/cpurc*.

Adicionalmente, para facilitar el traslado de programas a plataformas Plan 9, se facilita una librería para importar aplicaciones *POSIX*, así como otra de emulación del mecanismo de sockets de Berkeley que se ha estudiado en esta serie.

## UN ESCENARIO DE EJEMPLO

Un sistema Plan 9 gira en torno a tres tipos de componentes:

1. Un conjunto de terminales de acceso
2. Un servidor de ficheros
3. Otros servidores: fundamentalmente, servidores de CPUs rápidas, servidores de autenticación y *gateways* con otras redes.

Como ejemplo de escenario en un entorno de grupo de trabajo universitario, por ejemplo, se podría imaginar una red con 5 máquinas 486 con 16 Mb de RAM actuando como terminales Plan 9, una SUN Sparcstation-2 con 128 Mb de RAM y 8 Gb de disco duro como servidor de ficheros, dos Pentium 166 con 64 Mb de RAM como servidores de CPUs, un 386 con 8 Mb de RAM como servidor de autenticación y otro 386 con 4 líneas serie como servidor de PPP para acceso telefónico. El sistema se adaptaría fácilmente pudiendo reutilizar el hardware que va quedando obsoleto. Así, si dentro de un tiempo el 486 queda obsoleto (del todo, es decir, aún más), podría pasar a emplearse como servidor de autenticación, con lo que seguiría siendo útil. De la misma forma que en este entorno de ejemplo, los 386 realizan una labor "estupenda". Si se adquiere una SGI Challenge o un futuro P7, el sistema se reconfiguraría y pasaría a ser un servidor de CPUs. En definitiva, la máquina virtual Plan 9 formada por el conjunto mejoraría al cambiar uno de sus elementos de la misma forma que un PC mejora al cambiar uno de sus componentes.

En cualquier caso éste es, simplemente, un ejemplo. Se recuerda al lector que el sistema puede funcionar desde un único 386 con 8Mb, aunque será en casos como el del ejemplo donde demustre sus ventajas frente a otros S.O. como UNIX.

## DÓNDE Y CÓMO CONSEGUIR PLAN 9

La distribucón completa de Plan 9 esta compuesta por dos libros, cuatro disquetes de 1.4 Mb y un CD-ROM. En los libros se pueden encontrar todas las páginas del manual así como otros documentos describiendo el sistema. En los cuatro disquetes se encuentra el kernel, el sistema de ventanas, soporte de internet y entornos de programación para C y Alef. El CD-ROM contiene todo lo demas, kernels, librerías y utilidades para todas las arquitecturas, así como las fuentes de todo el sistema.

Dado que Plan 9 no emplea la BIOS y, por tanto, podría no funcionar en todos los PCs, la distribución presente en los 4 disquetes es gratuita para evaluación y se puede conseguir en:

*ftp://plan9.bell-labs.com/plan9/pcdist*

La distribución de la versión completa la realiza *Harcout Brace and Company* y para adquirirla desde España hay que llamar al 07 1 407 345 3800, o enviar un fax al 07 1 407 345 4060. El precio de la versión completa es de $350 y la referencia ISBN 0-03-017143-1. También es posible adquirir los manuales independientemente al precio de $125 cuya referencia ISBN es 0-03-017142-3. A los precios de las distribuciones añadirán los gastos de envío. Teóricamente si se solicita el número de ISBN a una librería, ésta podría realizar el pedido.

## FUTURO: BRAZIL, INFERNO Y LIMBO

En el momento de escribir este artículo, se emplea Plan 9 como S.O. en diversas universidades en Toronto, York, Sydney y Austria, ya no como fin en sí mismo, sino como soporte para el desarrollo de sistemas de ficheros, algoritmos de tratamiento de colas, desarrollos multimedia y programación diversa. En AT&T, más de 40 desarrolladores de Bell Labs emplean una evolución de Plan 9, de momento de uso interno, denominada *Brazil* como entorno de trabajo. Brazil pretende ser el siguiente paso en la evolución de Plan 9 que se logra mediante la reestructuración de la ruta de datos del kernel. Se trata de una optimización para máquinas y redes de altas prestaciones, ya que se está empleando en una red de fibra optica conmutada de 300 Mb/s de ancho de banda. El objetivo primordial es eliminar todo el *overhead* innecesario de Plan 9 eliminando todas las copias de datos que sea posible en la comunicación.

También se ha sustituído completamente el mecanismo de gráficos, que en Brazil adopta un modelo basado en datos. El sistema transmite actualizaciones de los datos a los servidores gráficos en lugar de las intrucciones sobre cómo dibujar las actualizaciones. De esta forma, es la aplicación y no el servidor de gráficos la que maneja directamente los datos que se muestran. El sistema se ha escrito en el nuevo lenguaje concurrente *Alef*, creado para el diseño de servidores multiproceso y basado en el paso de mensajes y memoria compartida.

El proyecto Brazil se encuentra en este momento en suspenso debido a que todo el equipo de trabajo ha sido destinado al proyecto *Inferno*, para el desarrollo de un nuevo sistema operativo de red con un entorno de desarrollo propio basado en el lenguaje *Limbo*, de similares características a *Java*, para el reparto de contenidos en un entorno de red rico en clientes y servidores heterogéneos. Inferno y Limbo serán tratados en un próximo número.

### BIBLIOGRAFÍA

Publicaciones impresas:
- Quentin Fennessy, "Plan 9: Bad Movie, Good Operating System". IEEE Computer, May 1996, P.117.

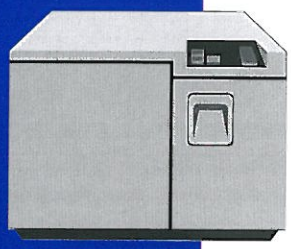
WWW:
- "An intro to Brazil", *http://plan9.wtf.nyc.ny.us/brazil.html*, resumen de una charla ofrecida por Rob Pike en la Universidad de Arizona el 23 de Enero de 1995.
- "Plan 9 form AT&T Labs", *http://plan9.bell-labs.com/plan9/index.html*
- Steve Kotsopulos, "Plan 9 from AT&T Bell Laboratories FAQ", *http://www.ecf.toronto.edu/plan9/plan9faq.html*
- Dennis Ritchie, "Plan 9 Q&A" (press release). *http://plan9.bell-labs.com/plan9/faq.html*

News:
*comp.os.plan9*

Mailing list: 9fans
Suscripción: *9fans-request@cse.psu.edu*
Archivo de mensajes: *http://plan9.wtf.nyc.ny.us/mindex.html*

# OBJETOS NATURAL (II)

*José María Peco*



**Como continuación al tema iniciado el pasado mes, el autor, con el presente artículo, expone lo que para él supone cada uno de los objetos que permiten la modularización en NATURAL, así como el uso que en su modesta opinión se debe hacer de cada uno de ellos**

El pasado mes, por razones de espacio sólo se pudieron exponer las características de los objetos tipo áreas de datos, programas y subrutinas. Por esta razón, antes de entrar a explicar cómo debe usarse cada uno de los tipos que componen este producto, se comienza con este punto, que es continuación del referenciado.

## 1.- COPYCODE

De este tipo de objeto sólo existe el fuente ya que, como indica su nombre, es un fragmento de código fuente que será incluido en el objeto que le referencia en el momento de la compilación.

Por sus características, este objeto se utiliza principalmente para activar teclas de función, controlar códigos de retorno o tratamientos de errores. Debe estar en la misma librería que el objeto que contiene el *INCLUDE*.

De este tipo, cabe comentar que las áreas de datos podrían considerarse como *copycodes* pues, al fin y al cabo, sólo contienen definiciones de variables. Esto no es totalmente cierto, ya que las áreas de datos permiten definir variables haciendo uso del editor especializado en este tipo de objetos, al tiempo que por el hecho de poder compilar un área de datos se permite efectuar una serie de validaciones previas al proceso de compilación del programa, para así incorporarse a éste posteriormente con el consiguiente ahorro de tiempo(1). En cambio, los objetos tipo *COPYCODE* no permiten realizar una compilación previa ya que, entre otras cosas, faltan las definiciones de formato de las variables.

## 2.- MAPA

Este objeto es invocado mediante la sentencia *INPUT*, y al ejecutarse presenta una pantalla que servirá de *interface* con el usuario para la captura y presentación de la información.

Este tipo puede incluir llamadas a otros dos tipos: *HELPMAPAS Y HELPROUTINES*, al tiempo que permite incluir reglas de proceso.

Para su creación es recomendable el uso del *LAYOUT*, ya que esto proporciona comodidad y eficiencia. Este es un campo que figura en los parámetros de creación del mapa y permite incluir en el mapa que se está inicializando las definiciones del mapa referenciado en el campo *LAYOUT* (figura 1), lo cual permite homogeneizar, entre otras cosas, el formato general o diseño de las pantallas de una instalación o aplicación, tales como cabeceras, PFs, etc, sin necesidad de estar dibujando cada una de éstas.

De este tipo de objeto cabe comentar que la propia arquitectura de NATURAL libera la memoria ocupada por el módulo llamado cuando se regresa al programa llamante por terminarse la ejecución de aquel. Por tanto, podría darse el caso de que al ejecutarse en el programa principal un *REINPUT* como consecuencia de una validación no superada, el mapa al que se regresa no se encuentre ya en el *Buffer POOL*, con lo que tendría que leerlo de nuevo desde el fichero del sistema NATURAL.

Por otra parte, como ya se ha dicho, este objeto puede incluir también reglas de proceso que se ejecutarán secuencialmente(2) por niveles una vez que se pulse *INTRO*, lo cual puede ser una gran ventaja o una gran desventaja, dependiendo del uso que se haga de las mismas.

- Ventaja: Puede usarse para eliminar validaciones, y en consecuencia código, del programa principal, tales como

formatos de fechas, campos obligatorios, etc.

- Desventaja: Pues, en el caso de que se incluyan procesos de elaboración de datos en estas reglas, elimina la claridad de un procedimiento secuencial a la hora de buscar errores.

## 3.- HELPROUTINE

Este tipo de objeto es de proceso, pues es una rutina que se invoca como información de ayuda desde un campo de un mapa. Debe haberse codificado utilizando el editor de programas, y se activa al escribir el carácter "?" en el campo.

La asignación de la ayuda se realiza cumplimentando el atributo HE, tal y como muestra la figura 2.

La figura 2 del pasado mes muestra el código de una HELPROUTINE, demostrando que es como una rutina normal, con la salvedad de que solo tiene en el área de parámetros como primer campo, el campo donde se encuentra definida la llamada.

## 4.- HELPTEXT O HELPMAPA

Es un mapa que se invoca como información de ayuda, y por tanto se ha codificado utilizando el editor de mapas. Se invoca exactamente igual que la *HELPROUTINE*, haciendo uso del atributo HE.

Se diferencia de la *HELPROUTINE* en que el *HELPTEXTO* es sólo un mapa con información sobre el campo, pero no recibe ni pasa parámetros, mientras que la *HELPROUTINE* es una rutina que normalmente lista datos para poder elegir un elemento.

## 5.- DESCRIPCIÓN

Este objeto recoge la descripción de un programa tal y como se encuentra definido en el diccionario de datos *PREDICT*. Sólo puede utilizarse este tipo si el sistema tiene instalado *PREDICT*.

## INTRODUCCION A LA 2ª PARTE

Los distintos objetos que pueden usarse bajo la plataforma NATURAL, hasta ahora han sido descritos en plan general, tal y como cuentan los manuales proporcionados por la casa diseñadora, *SOFTWARE AG*, para que sirva de introducción a esta segunda parte, con la cual el autor expone una visión particular del tema de acuerdo con su experiencia, al tiempo que define cómo se pueden combinar los distintos objetos a la hora de diseñar un proceso. No obstante, tampoco descubre nada que en teoría no se sepa, ya que simplemente se limita a considerar al "programa" como un elemento transformador de información, e incorpora a su planteamiento el principio básico de que todo programa debe tener un único punto de entrada y un único punto de salida, pues en informática también es aplicable el dicho popular "casa con dos puertas(3) mala es de guardar".

Figura 1

```
15:10:19          Define Map Settings for MAP                96-04-18

 Delimiters                Format                    Context
 ----------                ------                    -------
 Cls Att CD  Del           Page Size ..... 23        Device Check .... _______
 T   D       BLANK         Line Size ..... 100       WRITE Statement  _
 T   I       ?             Column Shift ... 0 (0/1)  INPUT Statement  X
 A   D                     Layout ........ _______   Help _______
 A   I       )              dynamic ..... N (Y/N)     as field default N (Y/N)
 A   N       ^             Zero Print ..... N (Y/N)
 M   D       &             Case Default ... UC (UC/LC)
 M   I       :             Manual Ship .... N (Y/N)  Automatic Rule Rank  1
 O   D       +             Decimal Char ... ,        Profile Name .... SYSPROF
 O   I       (             Standard Keys .. Y (Y/N)
                           Justification .. L (L/R)  Filler Characters
                           Print Mode ..... __       ----------------
                                                     Optional, Partial .... _
                           Control Var .... _______  Required, Partial .... _
                                                     Optional, Complete ... _
                                                     Required, Complete ... _

Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help        Exit                                                  Let
```

## ESQUEMA GENERAL DE UN COMANDO

Hasta ahora se ha hablado de objetos Natural de una forma indeterminada, pero en cualquier Centro de Proceso de Datos (CPD) se debe distinguir entre procesos *Online* y procesos *Batch*. Ambos procesos pueden combinar todos los tipos de objetos descritos, pero el planteamiento que se haga de los mismos debe ser distinto, ya que, en el primer caso, se asocia a procesos muy cortos que tratan una única entrada o registro, mientras que los segundos tratan todos los registros de un fichero, y además los procesos que tienen asociados suelen ser más largos y pesados(4).

El esquema que se presenta a continuación permite dar vida a la regla "un programa, un mapa" (5) válido para procesos *Online*. Esta regla permite dar claridad al programa y facilitar el mantenimiento, pues obliga a descomponer en funciones elementales problemas complejos. Además, permite clasificar como proceso tipo programa todo aquel módulo que invoca a un mapa, dejando los tipos *subroutine* y subprograma para aquellos subprocesos que validan o elaboran información elemental. No obstante, se debe puntualizar que no todos los programas de tipo principal tienen que tener mapa, tal es el caso de los programas *batch*.

NATURAL, como ya se ha dicho, permite descomponer los farragosos

Figura 2

```
Arr W-COD-ERROR                                                    Fmt N4
--------------------------------------------------------------------------------
AD= OILT_____   ZP= OFF    SG= OFF   HE= 'LISOBJOH'_____________   Rls 0
NL= _____       CD= __     CV= ____________________________________  Mod Data
PM= __                     DY=
MM= ____________________________________

001   --010---+----+----+---020---+----+----+---050---+----+----+---070---+----
(XXXXXXXXXXXXXXXXXXXXXXXX  (XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX  (XXXXXXX
(XXXXXXXXXXXXXXXXXXXXXXXX  (XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX  (XXXXXXX
--------------------------------------------------------------------------------
SELEC      REFERENCIA          DIV          IMPORTE          TIPO   FECHA    ERR.
------ -------------------- --- -------------------- ------ -------- ----
>:X  (MMMMMMMMMMMMMMMMMMMM (XXX (MMMMMMMMMMMMMMMMMMMM (XXXXXX (XXXXXXXX.N999
>:X  (MMMMMMMMMMMMMMMMMMMM (XXX (MMMMMMMMMMMMMMMMMMMM (XXXXXX (XXXXXXXX (9999
>:X  (MMMMMMMMMMMMMMMMMMMM (XXX (MMMMMMMMMMMMMMMMMMMM (XXXXXX (XXXXXXXX (9999
>:X  (MMMMMMMMMMMMMMMMMMMM (XXX (MMMMMMMMMMMMMMMMMMMM (XXXXXX (XXXXXXXX (9999
>:X  (MMMMMMMMMMMMMMMMMMMM (XXX (MMMMMMMMMMMMMMMMMMMM (XXXXXX (XXXXXXXX (9999
>:X  (MMMMMMMMMMMMMMMMMMMM (XXX (MMMMMMMMMMMMMMMMMMMM (XXXXXX (XXXXXXXX (9999
>:X  (MMMMMMMMMMMMMMMMMMMM (XXX (MMMMMMMMMMMMMMMMMMMM (XXXXXX (XXXXXXXX (9999
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      HELP  Mset  Exit  <---  --->  --   -    +          <    >    Let
```

programas COBOL que existían antes de su llegada en el año 85 en programas que escasamente pueden ocupar dos pantallas, siguiendo en cualquier caso los seis pasos siguientes. El orden de los puntos es invariable, aunque puede darse el caso de que alguno de ellos se encuentre vacío:

1) Definición de datos.
2) Recepción de los parámetros iniciales.
3) Preparación inicial de datos.
4) Presentación de un mapa con los datos iniciales, que sirva de interface para el tratamiento de la información.
5) Validación de los datos especificados en el paso anterior.
6) Tratamiento de los datos.

En procesos *batch* éste esquema se puede modificar anteponiendo otro programa, de modo que el programa principal se desdoble en:

- Un primer programa encargado de seleccionar los registros a tratar.
- Un segundo programa, invocado normalmente vía *FETCH RETURN* o *CALLNAT*, que será el encargado de procesar un único registro de los seleccionados por el programa inicial.

A modo de ejemplo se va a seguir la utilidad que acompañó el pasado mes al artículo, por ser una aplicación de uso general y además ser completa, sencilla y no necesitar especificaciones, ya que se limita a listar un fuente desde el fichero ADABAS: *FUSER* donde el sistema guarda los distintos módulos. La clave de acceso a este fichero tiene una longitud de 18 posiciones, de las que las 8 primeras se redefinen como aplicación o librería, las 8 siguientes como objeto y las 2 siguientes se reservan como secuenciador para aquellos módulos largos.

## DEFINICION DE DATOS

Las áreas de datos deben especificarse para contener sólo las definiciones de aquellas variables que se usarán dentro de una cadena o transacción, y no para contener definiciones generales, que amplían innecesariamente la memoria a usar por los módulos que contengan dicha área. En este punto cabe recordar los comentarios y las recomendaciones hechas el pasado mes al hablar de las características de las Áreas de Datos, sobre todo en lo relativo a las *views* internas.

Figura 3

```
UTILIDAD JMPDES  <<<<<<< Utilidad para desarrollo >>>>>>>   13/05/96 18:11:20
                        Libreria: UTILIDAD Objeto: LISPGM0P
------------------------------------------------------------------------------
 0860       SET KEY PF5          NAMED 'MENU'
 0870       INPUT NO ERASE (AD=I HE='LISPGM0H') MARK 2 AND SOUND ALARM
 0880          08/19 '=================================='
 0890          09/19 '*   <<<<<< LISTAR PROGRAMA >>>>>  LOP  *'
 0900          10/19 '=================================='
 0910          11/19 '*                                    *'
 0920          12/19 '*  LIBRERIA :' *OUTIN W-LIBRERIA-AC
 0930                    (HE='LISPGM1H')  '                *'
 0940          13/19 '*  OBJETO   :' *OUTIN W-PROGRAMA-AC
 0950                    (HE='LISPGM2H',W-LIBRERIA-AC) '                    *'
 0960          14/19 '*                                    *'
 0970          15/19 '=================================='
 0980          16/19 '* <DEJE TODO A BLANCO PARA IR AL MENU> *'
 0990          17/19 '=================================='
 1000       IF *PF-KEY = 'PF3'

      Proxima pantalla listar desde  1010     Buscar: _____________________
------------------------------------------------------------------------------
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      AYUDA       SALIR LOCAL MAPA  PRINT OBJ   VIEW  EXPLO <<---- --->>
```

Por otra parte, y siempre con las miras de dar claridad al mantenimiento(6), es muy útil utilizar distintos prefijos para referenciar las variables. Así, a modo de ejemplo, se exponen los siguientes:

P- para asociar a las variables que se reciben como parámetros.
W- para asociar a las variables de trabajo (working).
T- para asociar a variables tipo tabla o array.
G- para asociar a variables del área de datos global.

Un error muy común consiste en incluir todos los campos de una vista en un área de datos local, ya que esto implica no sólo un aumento de la memoria a usar, sino que además complica la labor de ADABAS por tener que crear más IFB (7).

## RECEPCION DE LOS PARAMETROS INICIALES

El siguiente punto del esquema definido para un objeto de tipo proceso es el que se refiere a la recepción de parámetros iniciales. Este punto es opcional, y puede hacerse de varias formas, dependiendo del tipo de ejecutable que se esté tratando. Evidentemente, en el caso de no recibir datos iniciales, cuando se ejecuta este tipo de objeto se debe examinar una situación, un tipo de registro, etc., ya que, por definición, un programa elabora información en base a una entrada.

En objetos tipo Programa sólo puede hacerse la recepción de parámetros ejecutando la sentencia *INPUT*. NATURAL, al encontrarse con esta sentencia, examina el área de usuario dedicada al *STACK* para ver si éste contiene datos. En caso de que así sea, no detiene la ejecución del programa, sino que asigna a cada una de las variables referenciadas en el *INPUT* los distintos valores que se encuentren apilados en el *STACK* (8).

La figura 3 muestra un *Hardcopy* con el *INPUT* directo que pide el nombre de librería y el nombre del fuente a listar. Al objeto de hacer completo el ejemplo, se ha incluido una llamada a una *HELPROUTINE* simple (sirve para seleccionar la librería entre todas las de la instalación) , y otra llamada a la *HELPROUTINE* que permite seleccionar un programa de entre todos los de una librería, por lo que se debe pasar también el nombre de la librería como parámetro.

En objetos tipo *Subrutina* los datos iniciales pueden recibirse vía *STACK*,

Figura 4

```
UTILIDAD JMPDES  <<<<<<< Utilidad para desarrollo >>>>>>>   14/05/96 09:27:18
                        Libreria: UTILIDAD Objeto: LISPGM2H
------------------------------------------------------------------------------
 0010 ************************************************************************
 0020 *
 0030 * APLICACION : INTERNACIONAL DESARROLLO
 0040 * PROGRAMA   : LISPGM2H
 0050 * OBJETIVO   : LISTAR PROGRAMAS DE UNA LIBRERIA
 0060 * AUTOR      : JMP
 0070 * FECHA      : 23/08/94
 0080 *
 0090 ************************************************************************
 0100 *
 0110 DEFINE DATA
 0120     PARAMETER 1 P-LIBRERIA (A8)
 0130               1 P-PROGRAMA (A8)
 0140     LOCAL
 0150 *

      Proxima pantalla listar desde  160_     Buscar: _____________________
------------------------------------------------------------------------------
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      AYUDA       SALIR LOCAL MAPA  PRINT OBJ   VIEW  EXPLO
```

```
UTILIDAD JMPDES   <<<<<<< Utilidad para desarrollo >>>>>>>   14/05/96 09:28:00
                          Libreria: UTILIDAD Objeto: LL_____
------------------------------------------------------------------------------
 0110 DEFINE DATA LOCAL
 0120      1 W-PROGRAMA (A8)
 0130 END-DEFINE
 0140 IF *DATA = 1
 0150    INPUT W-PROGRAMA
 0160    RELEASE STACK
 0170 END-IF
 0180 IF W-PROGRAMA = ' '
 0190    MOVE '*' TO W-PROGRAMA
 0200 END-IF
 0210 STACK TOP DATA 'UTILIDAD' W-PROGRAMA
 0220 FETCH 'LISPGM0P'
 0230 END

      Proxima pantalla listar desde  0100     Buscar: _____________________
------------------------------------------------------------------------------
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      AYUDA       SALIR LOCAL MAPA  PRINT OBJ   VIEW  EXPLO
```

Figura 5

igual que en los de tipo programa, o definiendo las distintas variables dentro de un área de datos *parameter*. En este caso no es necesario ejecutar ninguna sentencia para asociar los distintos valores a las variables ahí definidas, ya que cuando comienza la ejecución del módulo los valores son cargados en las correspondientes variables. En cualquier caso, los valores pasan del modulo llamante al modulo llamado de la siguiente forma:

*Perform Subrutina var_1 var_2 ...*

En Objetos tipo Subprograma se pueden hacer las mismas consideraciones que en los objetos tipo *subroutine*. No obstante, este tipo de objetos tiene la particularidad de no compartir los datos de las áreas globales(9) ya que, en principio, este tipo de objetos se crea para descentralizar la ejecución de programas, permitiendo incluso que se corra el programa principal en una máquina y el subprograma en otra. Este tipo de objetos normalmente es usado para definir funciones muy concretas y sencillas, que reciben datos y devuelven otros datos elaborados. La sentencia que se usa para invocar un subprograma tiene la forma siguiente:

*Callnat Subprograma param_1 param_2 ...*

La figura 4 muestra un *Hardcopy* con la definición del área de parámetros correspondiente a la *HELPROUTINE* que ayuda a seleccionar el nombre del fuente.

La variable del sistema **data* permite consultar el número de entradas que se encuentran en la pila de datos del área de usuario. Esta variable puede consultarse por programa como cualquier otra variable, y en caso de que devuelva el valor -1 significa que el *stack* contiene un comando en lugar de un dato.

La figura 5 muestra el listado del programa correspondiente al comando LL(10). Este comando sólo es un nombre de programa abreviado que permite invocar al programa raíz de la utilidad, pero escribiendo menos.

Es de resaltar el hecho de que la pila o *STACK* sólo contiene valores, no nombres de variables (continentes).

En este apartado se puede comentar que en objetos tipo programa y subrutina también puede recibirse datos utilizando el GDA(Área de Datos Global) para recoger datos del programa llamante. Pero no es aconsejable, ya que elimina la claridad de mantenimiento del objeto, pues se requerirá un estudio de todo el programa para conocer cuales son los datos iniciales que sirven de parámetros.

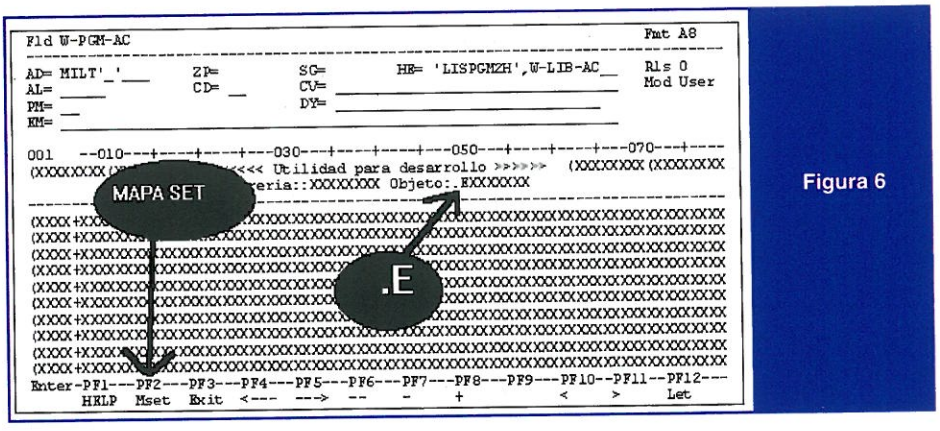



Figura 6

## PREPARACION INICIAL DE DATOS

Este punto del esquema, opcional, no suele ser un punto complejo, pues simplemente tiene por objeto completar, en base a los parámetros iniciales, los datos que se presentarán al usuario para su tratamiento en el punto siguiente del esquema.

Los objetos externos que puede incluir este punto son generalmente llamadas para descodificar algún valor, cosa que puede hacerse vía *Perform* o *CALLNAT*, para así poder presentar estos valores en el siguiente apartado.

Para el caso actual, en el programa *LISPGM0P*, este punto se corresponde con la rutina encargada de determinar el tipo de objeto al que pertenece el fuente, y en el programa *LISPGM1P* este punto se corresponde con la rutina que carga la tabla de números de línea y de texto.

## PRESENTACION DE UN MAPA CON LOS DATOS INICIALES, QUE SIRVA DE INTERFACE PARA EL TRATAMIENTO DE LA INFORMACION

Este punto del esquema suele invocar a otro objeto ejecutable de tipo MAPA mediante la siguiente sentencia:

*INPUT using MAP nombre_mapa*

Es un tipo especial, en el sentido de que éste tipo, además de servir de interface ante el usuario , es la cara por la que el usuario va a conocer la aplicación. Este objeto está adornado con todo tipo de facilidades, como ya se ha visto, tales como reglas de proceso, helprutinas y helptextos.

La figura 6 muestra cómo asociar la *HELPROUTINE* (11) que permite seleccionar un fuente, de entre todos los de una librería, en un mapa haciendo uso del atributo HE.

Antes de invocar al mapa se deben definir las Pfs que se usaran, o desactivaran, etc... E inmediatamente después de invocarle se puede incluir el proceso de examinar la PF pulsada por el usuario

a fin de eliminar validaciones innecesarias en el caso de que este decida cancelar el proceso.

Para asociar un mapa de ayuda a un mapa concreto, se especifica éste en el campo *HELP* del mapa *set* de la figura 1. Y para activar la llamada a este mapa debe programarse una PF (normalmente la PF1) con la palabra *HELP* (sin encerrar entre comillas), tal y como se indica a continuación:

*SET KEY PF1 = HELP NAMED 'AYUDA'*

De esta forma, dicho mapa de ayuda es invocado cuando se pulse dicha PF.

La figura 7 muestra cómo el tipo de objeto MAPA cuenta con un área de parámetros, por lo que el paso de datos al mapa puede hacerse de dos formas:

a) Forma implícita: sin enumerar los parámetros a pasar. En este caso todos los nombres de las variables del área de parámetros del mapa deben existir y coincidir en nombre y formato con los del módulo llamante.

b) Forma explícita: Enumerando a continuación del nombre del mapa en el programa llamante todas las variables del área de parámetros del mapa. En este caso, sólo deben coincidir en el orden los formatos de las variables pasadas.

Otra consideración importante es la que se refiere a la ubicación del mapa. Este debe figurar en el programa principal, y no dentro de una subrutina interna ya que, de hacerlo así, cuando se ejecute la sentencia REINPUT (12) por no haber superado una validación, si ya se ha salido de la subrutina que contiene el mapa, el sistema devuelve un error por no saber cómo acceder a él.

En el ejemplo, el programa LISPGM0P, este punto se corresponde con la secuencia vacía.

## VALIDACION DE LOS DATOS ESPECIFICADOS EN EL PASO ANTERIOR

Una vez captada la información mediante el mapa, y antes de iniciar la elaboración de la misma, se debe efectuar la validación de datos, a fin de evitar el deshacer todos los pasos realizados por culpa de no superar una validación tardía.

En este sentido, se deben tener presentes dos instrucciones muy importantes en este entorno:

- *REINPUT*: retorna el flujo de proceso al último *INPUT* ejecutado, que en el caso de que siga el esquema planteado sería el mapa, pero no deshace los cambios efectuados en los valores de las variables. Es decir, si se debe actualizar algún contador, esta actualización debe ejecutarse una vez que se hayan superado todas las validaciones, pues en caso contrario su valor sería engañoso.
- *BACKOUT TRANSACTION*: deshace todos los cambios realizados en la base de datos y libera todos los registros que se encuentren en *HOLD* (13). No obstante, ésta sentencia no debería figurar en esta parte del programa, pues de acuerdo con el primer párrafo, sólo se validan datos, no se actualizan.

## PROXIMO MES

Una vez más, razones de espacio obligan a dejar para el próximo mes los siguientes puntos, con los cuales se cerrará el tema:

- Tratamiento de los datos.
- Procesos que necesitan pantalla de confirmación
- Los Objetos NATURAL y la POO
- Diagrama de Flujo
- Buffers de Natural

## UTILIDAD

La utilidad que acompaña a este artículo complementa la dada el pasado mes. Es una herramienta muy fácil de desarrollar, prácticamente los fuentes vienen distribuidos con el disco que acompaña a la revista, y sólo hay que sustituir el nombre de la *view SYSTEM-FUSER* por el nombre de la DDM que apunta al fichero *FUSER* del sistema, que en principio coincidirá con el de la utilidad.

Las ventajas son:

- Lista cualquier fuente de cualquier librería, saltándose el control de *NATURAL-SECURITY*
- Permite buscar cadenas dentro de fuentes.
- Con PF7 se relacionan los objetos referenciados en un objeto.
- Su autor la califica como herramienta imprescindible en cualquier centro de desarrollo por su potencia, sencillez de uso, y sobre todo para funciones de mantenimiento.

El Fichero LISPGM.NAT incluido en el disquete que acompaña a la revista contiene el listado formateado de los distintos fuentes de la utilidad.

## NOTAS MARGINALES

1) Se recuerda en este sentido que los programas NATURAL no son ejecutables en el sentido amplio de la palabra, sino que sólo pueden ejecutarse bajo la plataforma NATURAL, por lo que estos ejecutables podrían considerarse como COMANDOS de un programa EJECUTABLE principal llamado NATURAL.

2) Para listar la secuencia de instrucciones sólo hay que ejecutar el comando *LIST M nombre_map*

3) En informática es habitual, aunque no académico, el que un programa pueda abandonarse por múltiples puntos. Por ejemplo usando la sentencia *FETCH*.

4) Se entiende por tratamiento pesado aquel que tiene un gran número de accesos a bases de datos y/o ficheros.

5) Esta regla ha sido llevada a la practica por el autor de este artículo, y ha demostrado que siempre es posible seguirla, aunque a veces es necesario realizar un buen análisis previo.

6) Esta idea, la de facilitar el mantenimiento, debe estar presente en cualquier desarrollo, ya que ambas funciones no suelen realizarse por las mismas personas.

7) IFB: *Internal Format Buffer* (ver primera parte del artículo).

8) Ver líneas de comentario previas al Primer *INPUT* del *LISPGM1P*.

9) En el caso de utilizar áreas globales dentro de un subprograma, ésta se inicializa cada vez que se invoca al subprograma.

10) Se ha optado por llamarle LL ya que L es la abreviatura del comando del sistema que lista un fuente.

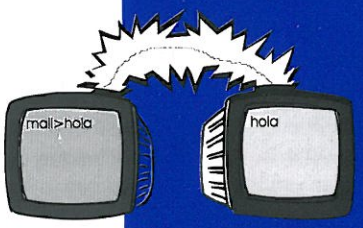
11) Para invocar a una *HELPROUTINE* no es suficiente con que el campo contenga una interrogación "?" sino que se debe escribir la interrogación y, estando el cursor sobre el campo en cuestión, pulsar la tecla *INTRO*.

12) La sentencia *REINPUT* cede el control del programa al último *INPUT* ejecutado.

13) Un registro se dice que se encuentra en *HOLD* cuando su lectura *puede suponer* una actualización del mismo, con independencia de que se cumplan o no las condiciones para su actualización. Cuando un registro se encuentra en este estado, no puede ser actualizado por otro usuario, devolviéndole a este ultimo el error 3145.

# SEGMENTACIÓN E INTERCONEXIÓN (II)

*María Jesús Recio*




En el articulo anterior se estudiaron diferentes elementos interconectantes de redes, pero se dejaron otros en el tintero. Además de los ya vistos, hubs y repetidores, existen otros dispositivos que trabajan a niveles superiores del modelo OSI y por tanto ofrecen unas prestaciones mayores.

Haciendo recuento de lo que se vio en la primera parte, sería bueno recordar el concepto de segmentación diciendo de él que es aquella *técnica* que permite dividir una red en subredes para mejorar el rendimiento de la misma; que la naturaleza de los motivos por los que se segmentaba una red provenía fundamentalmente de dos aspectos: la ampliación de una red, y el aumento del tráfico en la misma, ya sea por el tipo de información (vídeo, sonido), o por el aumento en sí de usuarios. Además habría que recordar el concepto de repetidor, como dispositivo que amplificaba y repetía una señal atenuada, y que se utiliza para aumentar la distancia entre dos hosts conectados; y el concepto de hub como dispositivo capaz de configurar una red con topología lógica en bus como una estrella física; que existen hub activos y pasivos y con diferentes número de puertos, y que a su vez varios hubs pueden interconectarse.

## MAU

MAU significa *Multistation Access Unit* (Unidad de acceso multiestación). Se trata básicamente de un hub que es capaz de implementar un anillo lógico (anillo colapsado) a través de una topología en estrella física. En la actualidad, cuando se habla de topología en anillo, se hace referencia no a la forma física del circuito que implementa el cable que forma la red, sino al circuito lógico que sigue un paquete transmitido por la misma, independientemente de la forma física de este circuito.

El término anillo colapsado es una individualización del término soporte colapsado que significa soporte reducido en tamaño y adaptado a una única caja; por ejemplo si tenemos una red con topología en bus, y el cable al que están conectados todos los ordenadores de la red, lo reducimos tanto de tamaño hasta ser capaces de meterlo en una caja a la que se conectan los ordenadores (mediante cables más largos, eso si), diremos que tenemos un bus colapsado. Un anillo colapsado, por tanto, puede implementarse a través de una MAU.

Ya se dijo que una topología física en anillo tenía algunas limitaciones, como por ejemplo, que si se rompía en algún punto del anillo el cable, toda la red quedaba paralizada; este era el motivo fundamental por el que físicamente se implementaba una estrella.

Obviamente, aunque una MAU sea un hub, y por tanto su apariencia física externa es muy similar, internamente no puede estar implementado de la misma forma. Mientras que un hub propiamente dicho, internamente tiene interconectados todos los puertos, para crear la apariencia de bus lineal, una MAU se configura en el interior como un anillo, como puede observarse en la figura 1.

Una MAU está capacitada para que cuando se rompe el anillo en algún punto, o bien alguna estación conectada a él falla (falla su tarjeta de red, por ejemplo) automáticamente desconecte de forma lógica la estación que falla o a la que está asociado el cable deteriorado y así mantener el anillo cerrado para que todo funcione correctamente.

**A veces no basta con un dispositivo que interconecte físicamente la red, se necesita algo más: Un dispositivo inteligente capaz de interconectar la red a un nivel superior y entender de encaminamientos y rutas. Especial importancia tiene éste mes el estudio sobre la interconexión de puentes.**

Para ampliar un anillo, basta conectar la estación a un puerto libre de la MAU; esta únicamente hace una extensión del anillo lógico, consiguiendo así ampliar el anillo con la nueva estación. Si no se tienen puertos libres, se debería pensar en la necesidad de conectar una nueva MAU, de forma que a la entrada de la primera MAU se conecte la salida de la segunda MAU y de esta forma se cierra el anillo (ver figuras 2a y 2b).

## ENLACE BACKBONE

Se trata de un cable que conecta entre sí dos o más segmentos de red local (no directamente sino a través de un dispositivo interconectante de segmentos) y ofrece un enlace de datos habitualmente de alta velocidad entre ellos, es decir, un enlace *backbone* permite interconectar segmentos de red manejados por un dispositivo como un bridge, un router, ya sean estos externos o creados a partir de la instalación de dos tarjetas de red en un servidor (siempre y cuando el servidor tenga funciones de gestión para ellas).

La figura 3.a muestra un enlace *backbone* entre segmentos de red, conectando directamente dos servidores a través de un cable de alta velocidad; ambos servidores implementan internamente un puente, gracias a la posesión de al menos dos tarjetas de red (una para el segmento en el que el servidor está situado y otra para conectar los segmentos restantes procedentes del cable de alta velocidad). Este tipo de instalación es muy común, cuando una red dispone de varios servidores.

Este tipo de enlaces se utiliza, entre otras cosas, para estructurar una red de forma que por ejemplo, si tenemos una red distribuida por un edificio, se puede crear un segmento de red por planta e interconectarlos a través de un cable de alta velocidad (enlace backbone) que pase por todas las plantas del edificio, como muestra la figura 3.b.

Otra de las aplicaciones para las que se utiliza un enlace *backbone* es para centralizar los servidores de una red, donde cada uno de ellos gestiona al menos un segmento de la red. Con esta configuración se interconectan los servidores a través de un enlace *backbone*. Las ventajas de centralizar los servidores son muchas, entre otras facilitar su mantenimiento, mejorar la seguridad física de estos, centralizar operaciones propias de los servidores; copias de seguridad, monitorización de la red, etc.

## WIRING HUBS

*Wiring hubs* o centralitas de cableado son centros de control que se utilizan para gestionar y monitorizar el cableado de una red esparcida por un edificio o campus. Se trata de un dispositivo con múltiples bases para módulos capaz de albergar dispositivos de diferentes tipos, hubs, bridges, por tanto podemos decir que una centralita de cableado es una plataforma para la interconexión de diferentes tipos de módulos de comunicaciones para redes.

Están compuestas por una fuente de alimentación (a veces dos, por si una de ellas falla) y un bus de expansión al que conectar diferentes dispositivos. Además suelen incorporar software adicional para la administración y monitorización de la red, ejecutable desde una estación.

## BRIDGE

Se trata de un dispositivo inteligente que es capaz de conectar dos segmentos de red, incluso si estos no son iguales, es decir, si tienen diferentes métodos de control de acceso al medio, por ejemplo uno pertenece al modelo 802.3 y otro al 802.5 (bus lineal y anillo).

Los puentes trabajan a nivel de enlace del modelo OSI. Esto significa que cualquier dispositivo que cumpla las especificaciones del subnivel MAC (que significa Media Access Control y que es el subnivel inferior en el que aparece dividió el nivel de enlace) puede conectarse con otros dispositivos que igualmente las cumplan gracias a un bridge. Al trabajar a este nivel, solamente entienden de direcciones MAC, y no de otro tipo de direcciones como pueden ser las direcciones IP, localizadas en el nivel de red, que es el que se encuentra por encima del nivel de enlace.

Siguiendo la similitud que en el primer articulo de esta serie se hizo entre el sistema de correo y las redes, se puede decir que un bridge es un clasificador que lee los paquetes de datos que llegan a él, estudia su dirección de red y los envía hacia la red (o subred) adecuada.

Los motivos por los que se puede instalar un puente son varios, y aparecen reflejados en la tabla 1.

Los puentes no saben nada de encaminamiento, es decir, saben las direcciones de las redes que interconectan y saben analizar la dirección de red de un paquete que les llega, enviándolo hacia la red destino por el camino que tiene establecido. No es capaz de estudiar el tráfico de la red en un momento determinado y elegir la ruta que debe seguir el paquete para llegar a su destino, en base a factores como tiempo, seguridad, fiabilidad, coste, etc. Aún así, un puente añade un tiempo de retraso en el paquete transmitido, puesto que tiene que analizarlo para sacar su dirección de red y redirigirlo así, a su destino.

Los puentes ofrecen funciones de filtrado, que posibilitan que el rendimiento de una red mejore cuando se segmenta a través de un puente. La función de filtrado consiste en no dejar pasar a segmentos de red paquetes, de información que no van destinados a ninguna estación de ellos.

A un puente le pueden llegar paquetes de diferentes naturalezas. La forma que tanto los puentes como las estaciones tienen para identificar el tipo de paquete que le llega: de información, de solicitud de ruta, de respuesta, etc.,

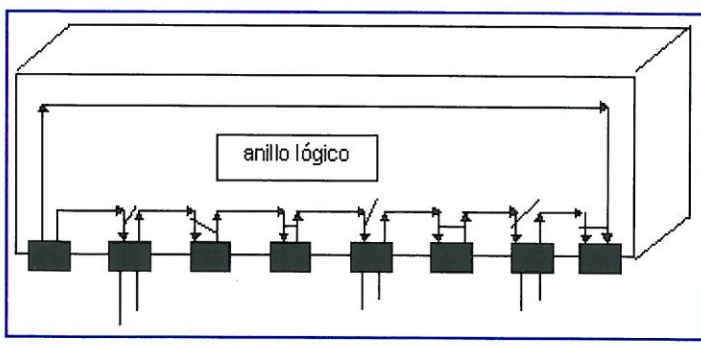



**Figura 1: Diseño interno de una MAU.**

se basa en la existencia de campos de control introducidos en los paquetes transmitidos.

Existen muchos tipos de puentes; puentes de aprendizaje, puentes tandem, puentes con distribución de carga, puentes transparentes, adaptativos, puentes de carga equilibrada, puentes locales, remotos etc. Las clasificaciones de los puentes se hacen atendiendo a diferentes aspectos. Además los puentes pueden materializarse como dispositivos externos, o bien a través de software, instalando varias tarjetas de red en un servidor, siempre y cuando dicho servidor contemple la posibilidad de implementar puentes, en el software que incorpora.

Una primera clasificación pudiera ser atendiendo al tipo de elementos que interconecta; los puentes locales son los que interconectan segmentos de una red local, o bien redes locales distribuidas como mucho sobre un edificio, mientras que los puentes remotos interconectan redes lejanas ente sí, a través de WAN, vía módem y sobre líneas de diferente naturaleza (analógica o digital).

Una segunda clasificación hace referencia al algoritmo seguido para localizar el destino de un paquete: puentes transparentes o adaptativos y puentes de encaminamiento fuente.

## PUENTES TRANSPARENTES O ADAPTATIVOS

Los puentes transparentes son más complejos que los puentes de encaminamiento fuente, pues deben ser capaces de deducir a partir de unas tablas de encaminamiento la dirección de la máquina destinataria del paquete recibido. Las tablas se van actualizando dinámicamente, añadiéndose en ellas nuevas direcciones. El puente asocia a cada puerta de entrada a él (correspondiente a un segmento) una tabla en la que apunta las direcciones de todos los paquetes que entran por dicha puerta; de esta forma genera una tabla por cada segmento de red. Cuando le llega un paquete por una puerta analiza si tiene apuntada su dirección; si no la tiene actualiza la tabla correspondiente a dicha puerta; a continuación analiza la dirección destino del paquete, y la busca en las tablas; si la encuentra envía el paquete por la puerta correspondiente a la tabla en la que la ha buscado, a excepción de si coincide con la puerta por la que llegó el paquete; en este caso supone que la trama llegó a su destino antes de llegar a él (el puente), por lo que descarta la trama; si no la encuentra, inicia un proceso de descubrimiento reteniendo el paquete de información que le ha llegado hasta que conozca el segmento de red por el que debe enviarlo) mandando un paquete de control (diferente al paquete de información que le ha llegado) a todos los segmentos de red, a excepción del segmento al que pertenece la máquina origen (la máquina que generó el paquete de información); cuando este paquete de control llega a la máquina destino, esta responde con otro paquete de control, que el puente, al recibirlo analiza, y deduce de esta forma el segmento de red al que pertenece la máquina destino pudiendo así enviar el paquete de información; además apunta esta nueva correspondencia para próximas ocasiones.

Uno puede pensar, que si se tiene una red muy grande, e incluso si se están conectando varias redes, el tamaño de estas tablas puede ser muy grande, por lo que la búsqueda en ellas ocasiona un tiempo de retardo en la transmisión del paquete considerable; existen diferentes métodos de optimización de las tablas, como puede ser el eliminar correspondencias que hace mucho tiempo que no son utilizadas, para así reducir el tamaño de las tablas. Además, los algoritmos de búsqueda en dichas tablas suelen estar implementados a través de hardware que siempre es más rápido. Un ejemplo de estas tablas aparece en la tabla 2.

Este tipo de puentes se suele utilizar para segmentar una red de tipo 802.3, y para interconectar redes 802.3 y redes en anillo.

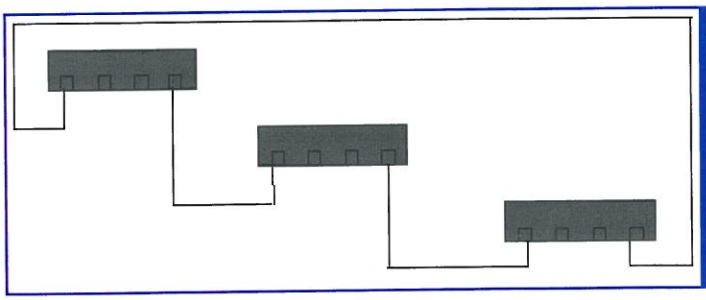


**Figura 2.a: Interconexión lógica de varias MAU's.**

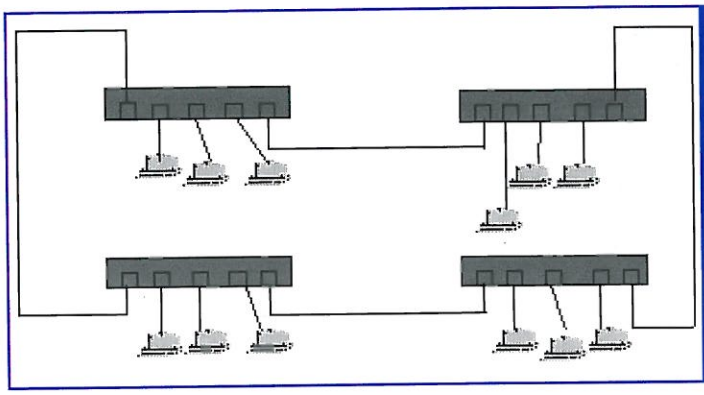


**Figura 2.c: Implementación de un anillo con varias MAU's interconectadas.**

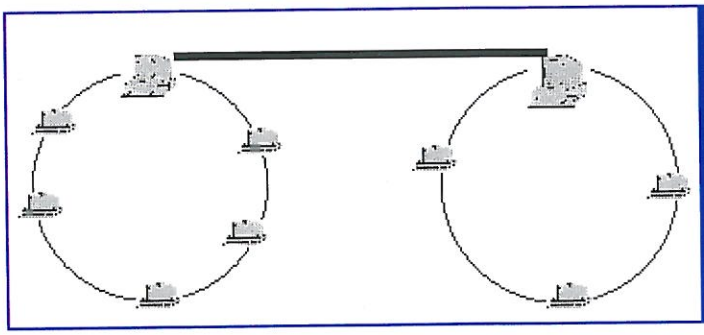


**Figura 3.a: Enlace backbone interconectando dos servidores.**

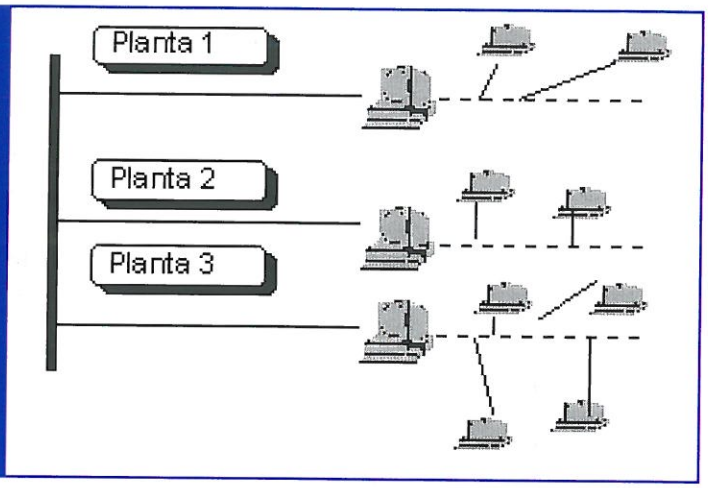



**Figura 3.b: Backbone estructurando la interconexión de segmentos de una red repartida por un edificio.**

## PUENTES DE ENCAMINAMIENTO FUENTE

Los otros tipos de puentes son los que emplean el algoritmo de encaminamiento fuente; estos puentes son más sencillos, ya que ellos solo deben leer la ruta descrita en el paquete para enviarlo por la puerta indicada.

Con este tipo de puentes, son las propias máquinas las que deben introducir en el paquete no solo la dirección destino sino además, la ruta que debe seguir para alcanzar su destino; por tanto, son las propias máquinas las de deben contener las tablas de encaminamiento. El modo de funcionamiento en este caso es el siguiente:

Cuando una estación quiere enviar un paquete a otra, busca la ruta para llegar a la estación destino en su tabla; si la encuentra, copia esta ruta en el paquete de información a transmitir y lo envía; si no la encuentra, entonces la estación origen envía un paquete de difusión para localizar a la estación destino; este paquete, en su viaje, va apuntando todos los puntos por los que pasa, es decir, va construyendo la ruta que se deberá seguir para llegar al destino.

Cuando este paquete llega a un puente se multiplica saliendo de esta forma por todos los segmentos o redes que interconecta. En algún momento, el paquete llega al destino, teniendo en cuenta que puede llegar más de una vez, (cuando se interconectan redes, pueden existir varios puntos de entrada a una de ellas, y por tanto, varios caminos por los que un paquete puede llegar de una red a otra). Sí existen varias rutas para alcanzar el destino, este analiza todas las rutas propuestas y selecciona la más idónea, en base a una serie de factores, descartando el resto . Una vez elegida la ruta reenvía el paquete de control con esta ruta, a la estación que lo originó, es decir, que hizo la petición. Al llegar al origen este extrae la ruta y la copia en el paquete de información que quería enviar. Además dicha estación apunta la ruta recibida en su tabla de encaminamiento para futuras transmisiones al mismo punto. Fíjese que es la estación destino la que selecciona la ruta, y no la estación origen del paquete.

Las ventajas que este método ofrece se centran fundamentalmente en la liberación de trabajo al puente, que únicamente tiene que leer la ruta y redirigir el paquete hacia el destino por el camino indicado; por tanto, los tiempos de retardo que introduce son menores.

Este tipo de puenteado es el que utilizan las redes Token-Ring para su segmentación, básicamente.

Los puentes tanto de encaminamiento fuente, como los transparentes, presentan una serie de inconvenientes:

- No diferencian entre clases de servicio (por el nivel al que trabajan), por lo que no son capaces de dar prioridad a unos paquetes sobre otros.
- No existe control de tráfico, por lo que un puente debe reenviar todos los paquetes que recibe o eliminarlos, pero no puede pedir a las estaciones que bajen el ritmo de transmisión de paquetes hacia él, por lo que el puente podría llegar a saturarse.
- El número de paquetes de control (de exploración y respuesta) aumenta el tráfico de la red, por lo que disminuye el rendimiento de la misma.

Además, como ya se ha dicho, los puentes pueden implementarse dentro de un servidor que sea capaz de gestionarlos, como pueden ser estaciones UNIX o servidores NOVELL. En estos casos, para implementar el puente se necesita instalar en el servidor tantas tarjetas de red como segmentos o redes se quieran conectar; evidentemente dichas tarjetas deben corresponder con la de los tipos de redes o segmentos que interconectan. Además, la máquina debe tener el software necesario para gestionar este puente.

En el caso de NOVELL (aunque con las mejoras introducidas en el software de gestión de los puentes, estos casi pueden considerarse como routers), el puente además de ser capaz de gestionar varios segmentos de red con funciones de filtrado, es capaz de, con una sola placa de red, y por tanto sobre un

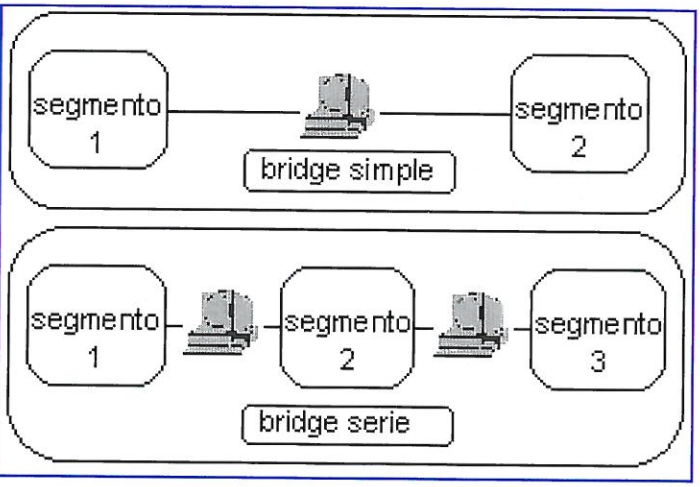



**Figura 4.a: Diferentes topología con bridges.**

único segmento, trabajar con diferentes protocolos (IPX/SPX, TCP/IP).

## INTERCONEXIÓN DE PUENTES

Existe un problema importante, cuando se quieren interconectar puentes transparentes con puentes de encaminamiento fuente, ya que las estaciones conectadas al puente transparente no entienden de tablas de encaminamiento.

Si queremos interconectar diferentes segmentos de red a través de puentes, podemos establecer diferentes topologías, dependiendo de características como: necesidad de evitar problemas en las conexiones, interconexión de varios segmentos distantes ,etc. Atendiendo a esto podemos clasificar las topologías con bridges en las siguientes: topología simple, paralela, serie, en bucle.

Una topología de bridge simple es aquella que conecta dos segmentos de red a través de un único bridge. La topología en paralelo se basa en conectar dos segmentos de red con dos bridges; los dos ofrecen las mismas conexiones, pero si uno de ellos falla el otro sigue haciendo posible la comunicación entre los dos segmentos; este tipo de topología se utiliza cuando se quiere tener mayor seguridad en las comunicaciones. La topología en serie conecta varios segmentos de red, intercalando entre cada dos segmentos un bridge, y por último, la topología en bucle se caracteriza por puentes conectados a segmentos formando entre todos un bucle cerrado; obviamente el número de segmentos mínimos en esta configuración es de tres. Se suelen utilizar fundamentalmente, con topologías en anillo. La figura 4.a. esquematiza las diferentes topologías.

**TABLA 1**

| |
|---|
| Extender una red, que ha alcanzado su número máximo de estaciones conectadas |
| Interconectar distintos tipos de redes. |
| Segmentar una red. |
| Eliminar cuellos de botella que se generan cuando existen muchas estaciones conectadas a un mismo segmento de red |

**Motivos para la instalación de un puente.**

## ROUTER

Los routers o encaminadores son dispositivos que trabajan a un nivel superior a los puentes, es decir, trabajan en el nivel de red, por tanto en el nivel en el que se encuentra el protocolo IP. Ofrecen, por tanto, mayores prestaciones que los puentes, ahora bien, también son más complejos que estos.

Un router interconecta varias redes, de forma que cuando un paquete tiene varios caminos para llegar a su destino, el router es capaz de seleccionar el mejor camino atendiendo a una serie de características (encaminamiento inteligente). El router selecciona el camino en base a unas tablas de encaminamiento; esta es una diferencia destacable con los puentes. Además de esta diferencia, podemos citar otras tan importantes como: capacidad de filtrado avanzado (aislar paquetes entre subredes), capacidad de manejo de varios protocolos (caso de routers multiprotocolo), etc.

Cuando un paquete llega a un router, este examina su dirección destino y comprueba sus tablas de encaminamiento, si puede enviar directamente el paquete a su destino lo hace, en otro caso determina la posición de otro router que puede enviar el paquete a su destino.

La elección del camino óptimo puede hacerla el responsable de la red, o dejársela al router. Los factores que se evalúan a la hora de seleccionar el camino óptimo son muchos, entre los que se pueden destacar: camino menos saturado (con menor tráfico), camino más rápido (por el medio de transmisión que lo implementa), camino más corto (por el número de saltos que el paquete debe realizar para llegar a su destino), camino más barato (cuando se utilizan líneas alquiladas).

Los routers pueden implantarse para el manejo de un solo protocolo o de varios (routers multiprotocolos). Un router para un único protocolo, por ejemplo IPX, solo puede manejar paquetes de ese protocolo, mientras que un router multiprotocolo puede gestionar paquetes de diferentes protocolos de forma simultánea.

Un router permite dividir una red en redes lógicas (segmentos de red); cada segmento tiene una dirección diferente a nivel de red (que en el caso de estar trabajando con IP a este nivel, sería una dirección de red IP diferente).

Estos protocolos definen el método por el cual los routers pueden comunicarse entre si; se ejecutan en el router construyendo tablas de encaminamiento basadas en el intercambio de información sobre caminos posibles. Existen diferentes protocolos, como son protocolos de vectores de distancia y protocolos de estado de enlaces.

Igual que en el caso de los puentes, se puede crear un router en una red NOVELL insertando dos tarjetas de red (una por segmento) en una estación el software correspondiente. No es necesario crear el router en el propio servidor NOVELL, se puede hacer en otra estación, y teniendo en cuenta que los routers deben procesar los paquetes y esto supone invertir tiempo, es preferible liberar al servidor de estar tareas.

## GATEWAYS (pasarelas)

Un gateway trabaja a nivel aplicación del modelo OSI. Un gateway propiamente dicho, es un dispositivo, que puede ser un ordenador que interconecta sistemas que no utilizan los mismos protocolos de comunicaciones, estructura de datos, lenguajes y arquitecturas, es decir, además de la función de encaminamiento que ofrecen los routers, incorporar una función de traducción entre protocolos; por tanto un gateway modifica el empaquetamiento de la información recibida y su sintaxis para adecuarse al sistema destino.

En el mundo Internet, actualmente se denomina gateway a aquel dispositivo que realiza funciones de traducción entre aplicaciones y no de encaminamiento, dejando esto último a los routers.

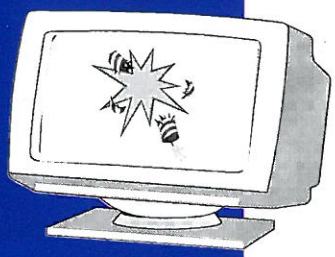
**TABLA 2**

| RED | DIRECCIÓN FUENTE |
|---|---|
| 10 | S01 |
| 10 | S02 |
| 10 | S03 |
| 11 | S08 |
| 10 | S09 |

**Ejemplo de tabla implementada en un puente transparente.**

# ALGORITMOS FRACTALES

*Pedro Antón*



**Este mes en Cómo Programar una demo se van a estudiar los algoritmos fractales, los cuales son muy utilizados para crear efectos vistosos de una manera muy sencilla. Descubramos todas sus posibilidades. Además se recuerda que hace tan sólo unos días se ha celebrado la Euskal Party**

El mundo de los fractales siempre ha sido admirado por todos, y la *demoscene* no va a ser menos. Existen *intros* (demos con un tamaño máximo de 4KB) que se hacen únicamente empleando algoritmos fractales. Esto es debido a la facilidad que éstos dan para crear efectos vistosos. Tengamos en cuenta que los fractales se usan para crear mundos de ciencia ficción o para implementar algoritmos de compresión de imágenes. Es decir, la utilidad que tienen en el mundo de los ordenadores es infinita y, por supuesto, queda mucho por descubrir.

En este artículo nos limitaremos a dibujar un par de superficies (ver figuras 1 y 2), que podrán ser usadas como fondo, textura o parte de una demo. No obstante hay que destacar que, en la mayoría de los casos, el código que acompaña a este artículo deberá ser optimizado a la hora de incluirse como parte de una demo, si se desea obtener un tiempo de generación de imagen aceptable.

costero, etc... no pueden ser definidas como simples figuras geométricas.

Beniot Mandelbrot fue el creador del termino fractal. Dicho término es un derivado de un adjetivo en Latín: *fractus*, que significa irregular, fragmentado.

El conjunto de Mandelbrot viene representado en la figura 1, y su algoritmo básico consiste en repetir para cada punto *c* el cuadrado de los numeros complejos más dicho punto c, comenzando por el punto 0. Si el valor obtenido excede de un límite suficientemente grande se termina la iteración. El color del punto será el numero de iteraciones realizadas. Dicho así resulta un tanto complejo, pero el lector observará en el Listado 1 que es relativamente sencillo de implementar.

**Algunas "intros" se hacen empleando únicamente algoritmos fractales**

## ¿QUE ES UN FRACTAL?

Antes de nada veamos algunas consideraciones sobre fractales.

Un fractal es una figura geométrica que puede ser subdividida en partes, las cuales, a su vez son aproximadamente un copia reducida de la totalidad de la figura. Existen muchas estructuras matemáticas fractales. Un ejemplo de ellas es el ejemplo que acompaña a este artículo: Mandelbrot. Pero en el mundo real también existen estructuras fractales: las nubes, las montañas, el relieve

Si en el conjunto de Mandelbrot en lugar de iterar para cada punto c, se itera el plano complejo se obtiene el conjunto de Julia, mostrado en la figura 2. La forma realmente rápida de conseguir realizar representaciones gráficas de este conjunto es realizar iteraciones inversas, ya que este conjunto converge más rápidamente de esta forma. Sin embargo para ilustrar la conexión entre ambos conjuntos se ha intentado que el código sea lo más similar posible, como se puede observar en el listado 2.

Se pueden escribir muchas líneas sobre los diversos aspectos matemáticos del conjunto de Mandelbrot, por no hablar de otras estructuras fractales, pero como ese no es el objetivo de esta artículo, aquellos lectores interesados en ese tipo de aspectos pueden consul-

**LISTADO 1**

```
Procedure MakeMandel;
Var
  XPos,YPos  : Word;
  RealP,ImagP : Real;
  CurrX,CurrY : Real;
  a2,b2       : Real;
  Cnt         : Byte;
Begin
  For YPos:=0 to (Height-1) do
    Begin
      For XPos:=0 to (Wide-1) do
        Begin

CurrX:=((XPos/Wide)*(MandelVal.XRMax-MandelVa
l.XRMin))
                    +MandelVal.XRMin;

CurrY:=((YPos/Height)*(MandelVal.YRMax-Mandel
Val.YRMin))
                    +MandelVal.YRMin;
          RealP:=0;
          ImagP:=0;
          Cnt:=0;
          Repeat
            a2:=Sqr(RealP);
            b2:=Sqr(ImagP);
            ImagP:=2*RealP*ImagP+CurrY;
            RealP:=a2-b2+CurrX;
            Inc(Cnt);
                  Until  (Cnt>=MandelVal.Colours)  or
(a2+b2>=MandelVal.Limit);
          PutPixel($A000,XPos,YPos,Cnt-1);
        End;
    End;
End;
```

tar las direcciones internet del final de este artículo, donde podrán encontrar las respuestas a diversos aspectos matemáticos, así como una extensa bibliografía donde conseguir extensa información sobre el tema.

## LOS FRACTALES Y LAS DEMOS

Este tipo de representaciones no podían pasar desapercibidas en el mundo de la *demoscene*. Algunos grupos han usado fractales para realizar una espectacular *intro* compuesta por multitud de paisajes fractales calculados en tiempo real. Otros sencillamente los usan como texturas, o como fondo o parte de una imagen. Los más osados se han atrevido a animarlos, aunque sobre este tema siempre ha existido una controversia: animar algunos conjuntos fractales en tiempo real es poco menos que imposible (aunque no es así con todos), y se ha recurrido a técnicas de semi-animación en tiempo real, realmente inapreciables para alguien que no sea muy experto en la materia.

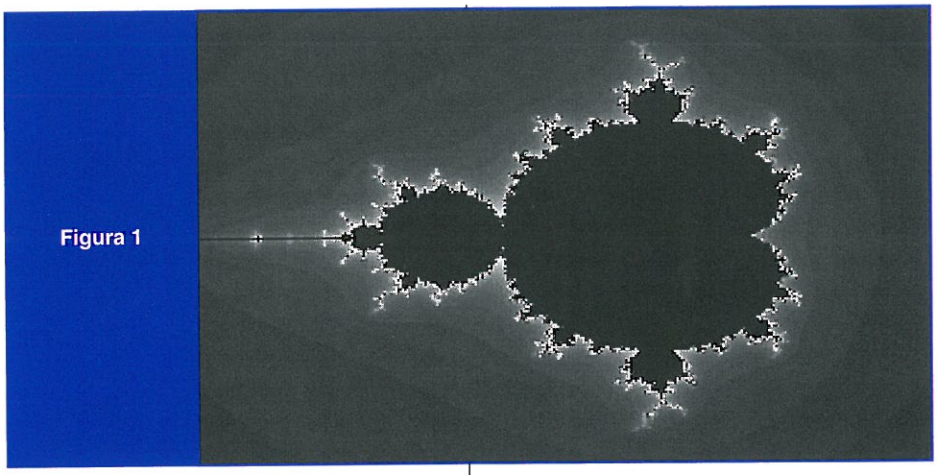


Figura 1

**LISTADO 2**

```
Procedure MakeJulia;
Var
  XPos,YPos  : Word;
  CurrX,CurrY : Real;
  a2,b2       : Real;
  Cnt         : Byte;
Begin
  For YPos:=0 to (LogicalHeight-1) do
    Begin
      For XPos:=0 to (LogicalWide-1) do
        Begin
          CurrX:=-2+XPos/(LogicalWide/JuliaVal.Limit);
          CurrY:= 2-YPos/(LogicalHeight/JuliaVal.Limit);
          Cnt := 0;
          Repeat
            a2:=Sqr(CurrX);
            b2:=Sqr(CurrY);
            CurrY := 2*CurrX*CurrY+JuliaVal.Complex;
            CurrX := a2-b2+JuliaVal.RealP;
            Inc (Cnt);
                    Until  (Cnt=JuliaVal.Colours)  or
(a2+b2>JuliaVal.Limit);
          PutPixel ($A000,XPos,YPos,Cnt);
        End;
    End;
End;
```

**LISTADO 3**

```
MandelDef = Record
            Colours : Byte;
            Limit   : Real;
            XRMin   : Real;
            XRMax   : Real;
            YRMin   : Real;
            YRMax   : Real;
          End;
JuliaDef  = Record
            Colours : Byte;
            Limit   : Real;
            RealP   : Real;
            Complex : Real;
          End;
```

## LA PALETA

Como suele ocurrir en la inmensa mayoría de los efectos de este curso, una bonita paleta dará otra imagen al mismo conjunto fractal, pero eso es algo a lo que ya estamos acostumbrados en este curso. Desde estas líneas, el autor quiere animar al lector a usar una paleta adecuada y hacer arder uno de los fractales del código que acompaña a este artículo, combinando dicho código con el primero de los artículos de este curso, el fuego.

**Se pueden realizar representaciones del conjunto de Mandelbrot rápidamente realizando iteraciones inversas**

## EL CÓDIGO

El código empleado es sencillo y claro, ya que la optimización de un algoritmo

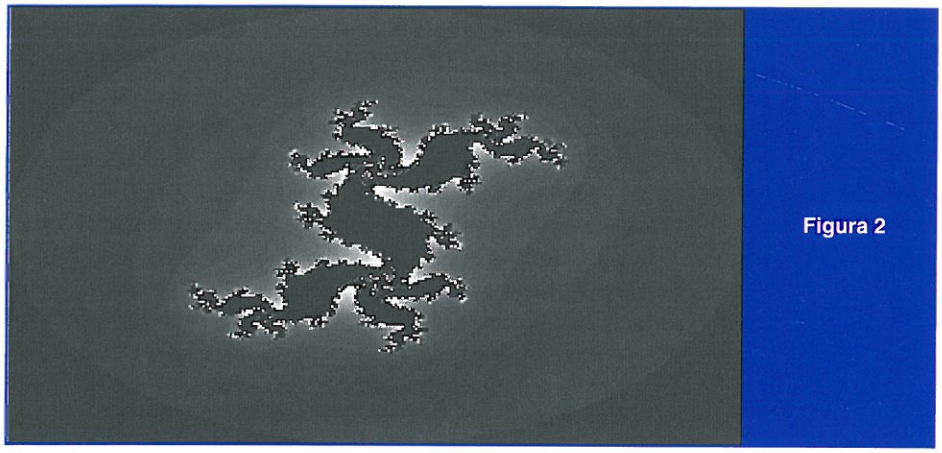


Figura 2

de representación de un conjunto fractal implica el uso, o bien de coma fija, o bien del coprocesador matemático. Todo ello, como puede suponer el lector, implementado en ensamblador, y el resultado no ilustra mucho esta explicación.

Las operaciones que se han de realizar para representar este tipo de imágenes requieren de gran precisión. Esto hace imprescindible el uso de variables de tipo Real en el código.

En el listado 3 se puede observar cómo la definición del subconjunto de Mandelbrot y el de Julia difieren en la zona a iterar. Mientras que en el primero se itera sobre una región, en el plano complejo, definida por los puntos complejos (XRMin,YRMin) y (XRMax,YRMax), en el segundo conjunto se itera sobre el punto complejo (RealP,Complex)

Y esto es todo por este mes, el lector podrá encontrar las respuestas a las preguntas mas comunes sobre el mundo de los fractales en internet consultando las páginas:

http://www.cis.ohio-state.edu/hypertext/faq/usenet/fractal-faq/faq.html.
http://www.marshall.edu/~stepp/fractal-faq/faq.html.

Y para finalizar, comentar para aquellos lectores que desconozcan el significado de las siglas *faq*, que su traducción literal es "cuestiones frecuentemente preguntadas" (*FREQUENTLY ASKED QUESTIONS*).

## EUSKAL PARTY

Hace tan solo unos días ha tenido lugar la tradicional party veraniega que todos los amantes de la demoscene no deben perderse, y por supuesto que Sólo Programadores asistió a tan importante evento, esperamos lo antes posible facilitar una relación de ganadores y nuestra opinión sobre sus trabajos.

Para los curiosos comentar que este año la Party ha tenido lugar en el frontón Carmelo Balda de San Sebastián, entre los días 25 y 28 de Julio.

**Para representar las imágenes es imprescindible utilizar variables de tipo Real**

A continuación se comentan algunas normas y categorías que han regido la Party este año ( los que no hayáis asistido nunca a un evento leerlas bien pues os pueden servir de ayuda para cuando os decidáis a participar)

- Máximo 4 megas en ejecución

- Deben poder ser detenidas con Esc en caso contrario descalificación

### AMIGA INTRO

- Tamaño máximo : 64 Kb.

- Deben poder pararse con el botón izquierdo del ratón, si no serán descalificadas durante las pruebas

### PC INTRO

- Tamaño máximo : 64 Kb.

- Deben poder pararse con ESC, si no serán descalificadas durante las pruebas, hasta que lo admitan.

### AMIGA INTRO 4Kb

- Tamaño máximo : 4096 bytes.

- Sí se permite que incluyan sonido.

### PC INTRO 4Kb

- Tamaño máximo : 4096 bytes.

- No se permite que incluyan sonido.

### MODS (Amiga y PC)

- En formato compatible Protracker (Se escucharan desde un Amiga).

- Sólo se escucharán 3 minutos.

- Tamaño máximo : 1 disco de baja densidad.

### MULTICANAL (Amiga y PC)

- Más de 4 canales.

- Formatos: S3M, XM, MOD(Fasttracker), Sunrize(Amiga), Tocatta (Amiga).

- Tamaño : 1 Mb.

- Se escucharán con una GUS estándar.

### GRÁFICOS 2D (Amiga y PC)

- Resolución máxima: 640 x 512 más overscan.

- Máximo número de colores: 256.

- Deberán estar hechos con un programa de 2D tipo Dpaint,Brilliance,etc.

- No se permitirá la inclusión de imágenes o brushes creados con programas de 3D o programas de retoque fotográfico como el photoshop o similares.

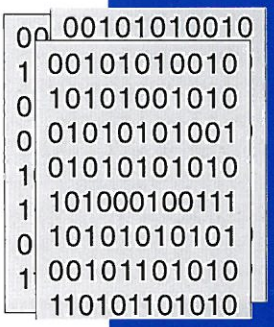
- No se admitirán digitalizaciones.

- Se mostraran en un Amiga.

- Formato: ILBM, TIFF, TARGA, GIF, PCX.

# TODO SOBRE LA BIOS DEL PC

*Enrique de Alarcón*




Si deseamos crear aplicaciones (sobre todo para DOS) que se adapten totalmente al sistema sobre el cual se van a ejecutar, debemos conocer todas las áreas de comunicación y todas las funciones que pone a nuestra disposición la BIOS. De esta manera, tendremos acceso a toda clase de información sobre cualquier cosa, como por ejemplo , el estado del teclado o el hardware del que disponemos (unidades de disco, puertos serie...) y a funciones tan básicas como la de obtener la memoria base instalada en el sistema o la de escribir en un sector del disco.

En este artículo se da una profunda explicación de todo el tema, con lo cual, el programador estará en condiciones de acceder a cualquier dato o servicio de la misma sin misterio alguno y con lo cual, podrá crear aplicaciones de mucho más bajo nivel.

## LA ROM-BIOS

BIOS son las siglas de *Basic Input/Output System* y que se traduce como "sistema básico de entrada/salida" y contiene tanto los microprogramas necesarios para que arranque tanto el diagnóstico del ordenador PC como una librería de funciones de unos 32Kbytes almacenadas en memoria ROM que ponen a disposición de cualquier programa (ya sea sistema operativo, aplicacion, juego ....) todas las operaciones básicas que se pueden realizar a bajo nivel, como es leer del disco, escribir en él, escribir en pantalla, etc... .Este conjunto de funciones es sobre el que se ha basado desde siempre el MS-DOS, y cuando por ejemplo, creamos un fichero con una función de interrupción del DOS, lo que estamos haciendo realmente es llamar a otras tantas de la BIOS (en este caso básicamente funciones de leer/escribir sectores).

Como hemos dicho antes, dicho código está registrado en memoria ROM , en forma para ser más precisos , de dos chips ROM (read only memory) llamados normalmente L y R. Como se puede deducir , es debido a eso mismo que su nombre completo sea ROM-BIOS. Además de estos dos chips , tenemos también la BIOS que traen consigo algunos componentes hardware , como por ejemplo , los discos duros y las placas graficas , que incluyen en sus propias tarjetas chips ROM con funciones para poder gestionarlos y que hacen las veces de ampliaciones de la BIOS de la placa base. Un ejemplo fácil de comprender es el Estándar Vesa para modos SuperVGA y que en algunos modelos de tarjeta gráfica , está incluido en forma de ROM en la propia BIOS para no tener que instalar un driver en el config.sys cada vez que se inicializa el sistema , lo cual ahorra memoria , pero imposibilita la actualización de dicho código.

## TIPOS DE ROM-BIOS

En general hay tres tipos de BIOS , la PC-BIOS , que es la básica y más elemental y que está instalada en los ordenadores más sencillos , la BIOS de los AT , que posee ampliaciones (más funciones) y la BIOS de los PS/2 , y la cual contiene 2 subtipos de BIOS debido a su mayor complejidad y que son las ABIOS y CBIOS. La ABIOS (de Advanced BIOS) contiene funciones para poder controlar características de modo protegido (código 32 bits) y la CBIOS (de Compatibility BIOS) posee funciones de ampliacion para el modo real.

## ÁREAS RAM USADAS POR LA BIOS

La ROM-BIOS , usa algunas pequeñas zonas de memoria RAM para almacenar en ellas datos de interés y variables del

**La BIOS del PC es una de las partes más importantes del sistema y sobre la cual funcionan casi todos los programas que se han realizado hasta ahora en DOS (inclusive el propio sistema operativo). En este artículo se explican todos sus secretos y además una completa lista de funciones BIOS básicas.**

sistema para que las aplicaciónes (incluyendo al DOS) puedan acceder a ellas. Existen 3 sub-áreas principales de comunicación:

*1-ZONA ROM-BIOS*, que está formada por la memoria comprendida entre los 640Kbytes y los 1024Kbytes y es donde se instala el código que de la BIOS. Las zonas libres que quedan en esta región se usan para almacenar controladores.

*2-ZONA CMOS:* Que son 64 bytes de memoria no volátil tanto de escritura como de lectura que está contenida en un chip RAM especial y que contiene toda la información que necesita la BIOS para poder ejecutar sus funciones correctamente. Así por ejemplo, de esta zona, extrae la BIOS la arquitectura física del disco duro (sectores, pistas...) para poder acceder al sector de arranque e inicializar el sistema operativo.

*3-ZONA RAM:* Zona que incluye la tabla de Vectores de interrupción, el area de comunicaciones de la BIOS y el area de comunicaciones de la BIOS extendida (las cual detallamos más adelante).

## LA BIOS Y EL ARRANQUE DEL PC

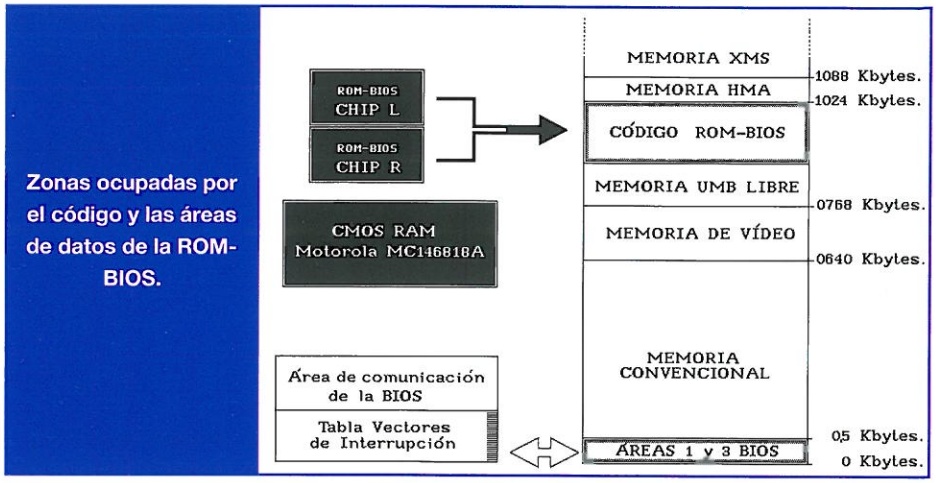
Una vez , encendemos el ordenador , lo primero que hace el microprograma de arranque de la BIOS es comprobar todos los componentes hardware que haya instalados en el PC. La lista de comprobaciones que realiza el sistema es el siguiente:

*-Test de Registro de la CPU:* Se compueba si los registros de la CPU retienen unos valores de prueba. En caso de no pasar este test , ya podemos ir pensando en cambiar la CPU.

*-Chequeo de la ROM-BIOS:* Se comprueba si el contenido de la ROM-BIOS tiene un CRC correcto, si hay resultado erróneo , finaliza la inicialización.

*-Chequeo del controlador de teclado:* El test es enviar un valor 0aah al controlador del teclado y recibir del mismo el valor 55h, si no devuelve dicho valor , se produce un error.

*-Chequeo de la CMOS:* Se comprueba si la memoria RAM CMOS retiene los valores que se le dan. Si no los retiene, da error (falta de baterías u otras causas).




*-Timer Test:* Realiza tests sobre los canales 0 a 2 del temporizador.

*-Refresco de la memoria:* Se comprueba si funciona correctamente la señal de refresco del temporizadorencargada de actualizar la memoria RAM para que no pierda su contenido.

*-Chequeo de 64Kbytes:* Se comprueba si los primeros 64 Kbytes de la memoria convencional funcionan bien (pruebas de escritura/lectura).

*-Comprobación de la caché:* Se compara leer 32 Kbytes de memoria con y sin caché. Si el tiempo necesario para dicha operación en los dos casos es idéntico , es que la caché no funciona bien , y se produce error. Para poder proseguir , deberá desabilitar la caché desde el setup y reiniciar el proceso.

*-Checksum de CMOS:* Comprueba si se ha modificado ilegalmente algún dato de la CMOS , comprobando el CRC actual con el salvado en la última actualización de los datos del SETUP.

*-Chequeo de Vídeo:* Comprueba que tipo de tarjeta gráfica está instalada (EGA,VGA...) y comprueba los primeros 4 Kbytes de la memoria de Vídeo que corresponda a la página 0 de vídeo a base de escribir y leer sobre ella.

*-Test Modo Protegido:* Como indica el nombre , se hace una prueba de pasar a modo protegido y volver a modo real para ver si funciona bien el cambio.

*-Memory Check:* Se comprueba si funciona correctamente toda la memoria base y extendida a base de operaciones de escritura/lectura sobre la misma.

*-DMA check:* Se comprueba si funcionan bien los canales DMA (Direct Memory Acces). a base de escribir valores en en los puertos correspondientes.

*-Setup de disquetera:* Chequea el estado de las unidades de disquete (por eso se encienden los leds al arrancar).

*-Setup de disco duro:* Comprueba el funcionamiento de los discos duros a base de resetear sus controladoras y de ejecutar los programas de diagnóstico que poseen las mismas.

*-Memory size check:* Comprueba si se detecta por BIOS la misma cantidad de memoria base y XMS que la registrada en la CMOS del sistema. En caso de no coincidir los valores , nos dará error.

*-Chequeo de ROM:* Se realizan comprobaciones complementarias del funcionamiento de la ROM-BIOS.

*-Initialize Timer data area:* Se comprueba el reloj interno , su funcionamiento y el estado de su batería. Cuando la batería esta baja , ocurren los molestos mensajes de reactualizar la hora y fecha.

*-Printer & RS232:* Se chequea la presencia de puertos de impresora y puertos RS-232 a base de enviar datos a puertos específicos. Aquellos puertos que hayan retenido los valores enviados , es que existen y se registrarán como válidos.

*-Keyboard lock:* Se examina si el teclado está bloqueado (si tiene la llave puesta).

*-Display errors:* Se sacan por pantalla todos los errores detectados en los tests anteriores.

*-System Rom Control:* Se comprueba el CRC de la ROM situada en el segmento E000h.

Esta es la última comprobación. A partir de aqui , se lee el sector de arranque y se pasa el control al sistema operativo.

- **Dirección 0040h:0089h:** 1 byte. Contiene en sus bits , flags de estado específicos de la VGA. Contenido: Bit 0= Estado de la VGA (1=activa) , Bit 1=Estado escala de gris (1=activada) , Bit 2=Tipo de monitor (0=color) , Bit 3= Estado paleta por defecto , Bit 4=Número de líneas de barrido , Bit 5=reservado , Bit 6=Estado cambio tarjeta , Bit 7= Complemento del bit 4 para número de líneas de barrido. El número de lineas es 350 , 400 o 200
- **Dirección 0040h:0090h:** 1 byte. Información sobre la disketera número 0 que esté instalada. Contenido: Bit 0-2=Tipo de diskette que se esta leyendo y en que tipo de unidad se esta leyendo. Así por ejemplo , 0 indica que se intenta leer un disco de 360Kbytes en una unidad de 360Kbytes. Bit 3=Reservado , Bit 4=Tipo de unidad de disco reconocida, Bit 5=Marca especial sobre funcionamiento de unidades de disco , bit 6-7= Velocidad de tranferencia de la unidad (entre 250 y 500 Kbytes/seg).
- **Dirección 0040h:0091h:** 1 byte. Tipo de la unidad de disco segunda que haya instalada . La información que contiene es idéntica en formato al anterior byte comentado.
- **Dirección 0040h:0096h:** 1 byte. Información sobre el teclado. Contenido de sus flags (1=activo): Bit 0=Si la última tecla pulsada fué el carácter 225 , Bit 1=Si el último carácter fue 224 , bit 2=Tecla CTRL , Bit 3=Tecla Alt derecha , Bit 4=Si existe conectado un teclado , Bit 5=Bloque Numérico , Bit 6=Desconocido , Bit 7=lectura en curso.
- **Dirección 0040h:00C0h:** 1 palabra. Días pasados en relación al 1-1-1980. Consiste en una especie de resta de fechas.
- **Dirección 0050h:0000h:** 1 byte. Resultado de intentar imprimir el contenido de la pantalla en impresora. Posibles errores: 00h= Si no ha habido error , 01h=Si está imprimiendo , FFh= Si ha habido error durante la impresión.

## EJEMPLOS:

Un ejemplo de código que usa un servicio BIOS sería por ejemplo , para formatear una pista , deberíamos llamar la función 13h subfunción AH=05h de la BIOS para gestión de discos , con el número de sectores en AL (1-9) , el número de la unidad en DL (0-3) , el número de cara en DH (0-1) y el número de pista en CH (0-39).Una vez realizada la INT 13h , en CF devolvería 1 si hubiese algún error y AH con el byte de estado. El código sería simple:

```
mov ah,05h
mov al,2
sub dl,dl
mov dh,1
mov ch,20
int 13h
jc _SI_ERROR
ret
_SI_ERROR:
; Gestión del error
```

Otro ejemplo muy sencillo y muy usado en todos los programas es el cambiar el modo de pantalla gráfico , para lo cual , tan sólo tenemos que usar la interrupción 10h con el servicio AH=00h y en AL el modo que queremos activar. El código es también muy simple:

```
sub ah,ah
mov al , _MODO_PANTALLA
int 10h
```

Como se ve , la utilidad y facilidad de uso de dichas funciones queda totalmente demostrada.

## LA SHADOW RAM

Los chips de memoria ROM de la BIOS , son una memoria muy lenta de acceder (leer) y además es de 8 bits, con lo cual , hace que las funciones BIOS no puedan usarse como funciones de alta velocidad , ya que se ejecución es muy lenta. Esta falta de velocidad , la podemos observar , por ejemplo , en una operación sencilla como la de llenar la pantalla de caracteres , realizando esta operación primero usando la función BIOS de escritura y después repetirlo escribiendo directamente en la memoria de vídeo.

Observando su lentitud , nos daremos cuenta de que la BIOS es casi una molestia al ejecutarse y por ello , se creó la llamada *SHADOW RAM* , que lo único que hace , es copiar algunos bloques de la ROM-BIOS al arrancar el ordenador en una zona de memoria RAM para que las funciones se ejecuten desde ahí y así eliminar el problema de velocidad de acceso. Esta diferencia de velocidad entre la ejecución desde ROM y RAM la podemos observar visualmente en la pantalla probando realizar operaciones desde MS-DOS (como por ejemplo un DIR ) activando y desactivando las opciones de SHADOW RAM desde el SETUP de la BIOS del sistema. En los SETUP's más modernos , se pueden incluso realizar copias de la BIOS en SHADOW RAM sólo de bloques específicos que nos interesen, pudiendo elegir entre copiar las funciones de disco , las de pantalla , u otras combinaciones. También debemos tener presente de que a más bloques copiemos en RAM , menos memoria libre tendremos como XMS (aun que no sea gran cantidad la pérdida).

## EXTENSIONES BIOS

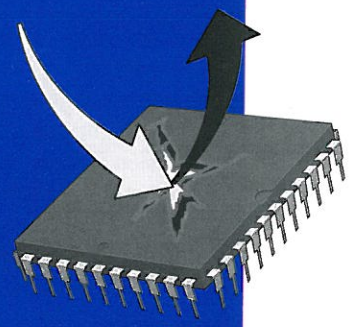
Como hemos dicho al principio del artículo , la BIOS que incorporan las propias tarjetas de expansión funciona como una ampliación de los servicios BIOS disponibles , como por ejemplo , el famoso estándar Vesa que traen en ROM algunas placas SVGA con funciones estándar para todos los modos SVGA , que se añade simplemente como una nueva interrupción XX , que al ser llamada , proporciona infinidad de nuevos servicios destinados , siguiendo el ejemplo , a controlar la tarjeta de vídeo , indicando el número de servicio como siempre en AH.

## RESUMEN

En este artículo se ha explicado la importancia que tiene conocer a fondo la BIOS , que es exactamente , cómo esta almacenada y cómo funciona.
Se ha detallado también los tipos de BIOS que podemos encontrar instalados según el modelo de PC que tengamos y las zonas que tiene asignadas en la RAM para comunicarse con el sistema. Después se han detallado todas las operaciones que realiza la BIOS durante el proceso de arranque (tema poco conocido normalmente) explicando por orden todos los tests y diagnósticos que realiza.

Una vez todos los conceptos han estado claros , se han listado todos los servicios de interrupción que pone a nuestra disposición la BIOS y detallado todos los datos que contiene la zona de comunicaciones. Para finalizar , se ha puesto algún ejemplo de como aplicar dichos servicios , detallado en que consiste la SHADOW RAM y explicado como se accede a las extensiones BIOS que incorporan los propios componentes hardware.

CONSTRUCCIÓN DE UN COMPILADOR (X)



# EL CÓDIGO DE CABECERA

*Daniel Navarro*

**Se termina el compilador. Los programas fuente en Letra se convierten ya en programas ejecutables, aunque aún no se ha visto más que una pequeña porción del mundo de los compiladores. En nuestro habitual CD-ROM todo el material para trabajar**

En este artículo se resolverán muchos detalles que alejaban hasta ahora al compilador de Letra de ser un compilador plenamente funcional. Ya se generaba código máquina ejecutable, pero faltaba algo muy importante: el código de la cabecera, o *start up code*, que traducido quiere decir algo así como "el código que despierta al programa".

Este código toma el control del sistema justo antes de que se ejecute el programa compilado, y cuando el programa finaliza, retoma el control antes de volver al sistema operativo. En Letra dicho código contiene además las funciones internas del lenguaje.

El *start up* se construye como si fuera un programa normal, que inicializa algunos aspectos básicos del sistema (como interrupciones) al comenzar y los restaura al salir, pero este programa no tiene "el cuerpo principal", ya que este cuerpo lo formará un programa compilado en Letra.

La mayoría de compiladores construyen su *start up* en lenguaje ensamblador, para Letra se ha construido también uno en ensamblador, pero como lo prometido es deuda, y se afirmó que no harían falta conocimientos de ensamblador para seguir el desarrollo del compilador, se ha construido otro *start up* equivalente en lenguaje C.

Escribir este código en C tiene un problema, y es que como básicamente se trata de un programa "normal", el compilador de C le añadirá su propio código *start up*. Debido a esto cualquier programa generado por el compilador de Letra, tendrá dos códigos *start up*: el escrito en C por nosotros, y el que el compilador de C le añade a este. Esto en caso de Letra no representa ningún problema serio, simplemente que el código *start up* construido de esta manera ocupará unas 5k (que se añadirán al principio de todos los programas en Letra), mientras que el equivalente escrito en ensamblador no llega a los 500 bytes.

¿Por qué no vale el propio *start up* de C para Letra?, pues sencillamente porque los programas en Letra tienen sus propios requisitos, como por ejemplo:

- Se debe comenzar la ejecución en el modo texto 80x25, con la pantalla borrada, y esto implica que es el código *start up* el que debe realizar dicha inicialización.

- Si se pulsa ESC durante la ejecución el programa debe salir al sistema operativo, lo que nuevamente implica que es el código de cabecera el que debe instalar una *interrupción de teclado* para que la tecla ESC realice dicha función.

- Al salir al sistema operativo se debe volver a borrar la pantalla.

- Además se debe inicializar el generador de números aleatorios, un temporizador en centésimas de segundo (con una *interrupción de reloj*), etc.

Las funciones que realiza el código de cabecera de C son otras diferentes, como inicialización de la pila, de la *heap* (memoria que el programa podrá pedir y liberar), vectores de interrupción (división por cero, desbordamiento

de pila, ...), variables de entorno, preparar los argumentos de main(), etc...

## COMUNICACIÓN CON EL COMPILADOR

El compilador y el código de cabecera deben comunicarse. El compilador debe comunicar al *start up* la dirección en la que comienza el programa compilado en Letra, para que el *start up* lo ejecute como si fuera su "cuerpo principal". Y el *start up* debe comunicar al compilador donde están las distintas funciones internas (para que el compilador pueda generar llamadas a las mismas) y cuál es la longitud total del *start up*.

El *start up* de Letra está contenido en el fichero "startup.com" (que proviene del fuente "startup.cpp"), este fichero será el que se añada a todos los programas compilados. El compilador cargará "startup.com" en el vector mem[ ] (a partir de la posición 256), y después generará código en el mismo vector, tras este código de cabecera.

Dentro de las variables del código de cabecera, la más relevante es la tabla de funciones (listado 1). En la primera posición de la tabla (funciones[0]) se indica la dirección del programa compilado que se añade al *start up* (que la definirá el compilador); esta será la dirección a la que el código de cabecera deberá llamar para invocar a su "cuerpo principal".

Cuando el compilador ejecuta "startup.com" directamente, sin haber indicado dicha dirección (y por tanto, sin haberle añadido un programa compilado al final), este guardará en dos variables de la BIOS la siguiente información (finalizando entonces su ejecución):

- La dirección en la que acaba el código de cabecera y por lo tanto está libre para generar código a partir de ella.
- La dirección en la que se encuentra la tabla de funciones dentro del código de cabecera.

El compilador ejecuta de este modo el código de cabecera (haciendo un system("startup.com")) y después lee estos dos valores. El primero le indica la dirección a partir de la cual va a generar código (el valor inicial de IP), y el segundo donde está la tabla con las direcciones de las funciones internas. Una vez conoce esta información, el compilador carga el *start up* en mem[ ] e indica en funciones[0] (ahora que sabe donde está la tabla de funciones) el valor inicial de IP (el comienzo del "cuerpo principal" del programa).

**Todos los programas compilados en Letra incluyen el código de las funciones internas, sean o no utilizadas**

Cuando se ejecuta el código de cabecera contenido en "startup.com" con un programa compilado añadido al final y con la dirección de inicio de este en funciones[0], entonces se llamará a la función de inicialización, al programa compilado y por último a la función de finalización. Es decir, todos los programas compilados en Letra no serán más que el "startup.com" con más código añadido al final.

## LLAMANDO A UNA FUNCIÓN INTERNA

Las funciones internas son funciones que están en el fuente "startup.cpp", es decir que están en el código de cabecera. Cuando el compilador necesita generar código para llamar a la función especial número "n", sabe que su dirección está en funciones[n], por lo que para generar una llamada a la función interna dentro del procedimiento de análisis correspondiente, sólo se tiene que hacer:

*generar(_lla,funciones[n])*

Si la función no tiene parámetros (como *borrar_pantalla()*) esto es suficiente para generar código. La función correspondiente dentro del código de cabecera (startup.cpp) se define como:

```
void borrar_pantalla(void){
  // Código...
}
```

Si la función tiene parámetros estos pueden ser de dos tipos:

- Por valor, cuando la función interna no va a devolver ningún resultado en el parámetro, en este caso el parámetro puede ser una expresión cualquiera (por ejemplo: *color(15)*).
- Por variable, cuando la función interna devuelve un resultado en el dato pasado como parámetro, en este caso el parámetro tiene que ser un identificador de dato o tabla (por ejemplo: *leer_tecla(n)*).

## PARÁMETROS POR VALOR

Este caso es el más sencillo, el procedimiento de análisis comprobará que la sintaxis de la llamada a la función interna sea correcta y generará código para evaluar la expresión que se va a pasar como parámetro (esto se hace automáticamente al invocar a *an_EXPRESION()*). Por tanto, al terminar el análisis de la sentencia se ha generado código para apilar el parámetro. Ahora se genera código para invocar a la función interna del mismo modo (suponiendo que vamos a invocar a *color()* que es la función número 5):

*generar(_lla,funciones[5])*

Pero la función correspondiente del código de cabecera se define ahora como:

**LISTADO 1**

```
void * funciones[ ]={
  &main,
  &borrar_pantalla,
  &escribir,
  &leer_tecla,
  &mover_cursor,
  &color,
  &aleatorio,
  &esperar,
  &poner_caracter,
  &leer_caracter
};
```

**Tabla de funciones.**

```
void color(int n){
  // Código...
}
```

Esta función en C tomará de forma automática el parámetro apilado como "n" (que podrá utilizar libremente dentro de su código), pero al retornar no lo desapilará, es decir, tras la llamada a la función el parámetro sigue apilado.

Por ello se debe sacar dicho valor de la pila, y como no había ninguna instrucción de EML que lo hiciera, se ha tenido que añadir una nueva, esta es ASP (de "add to stack pointer") que como parámetro requiere el número de bytes que se han de desapilar. Como cada parámetro ocupa 2 bytes, en el ejemplo de la función anterior (*color()*) que tenía un sólo parámetro, se generará código para desapilarlo con la siguiente instrucción:

```
generar(_asp,2)
```

La cadena 80x86 que implementa ASP y la instrucción ensamblador equivalente son las siguientes:

***ASP(2):*** *81 c4 00 00*
***ASP:*** *add sp,0*

Cuando la función tiene más de un parámetro, como por ejemplo *mover_cursor (x,y)*, se generará primero código para apilar "x" y después código para apilar "y". Los parámetros se apilan en Letra de izquierda a derecha, pero en el lenguaje C se apilan de derecha a izquierda, esto quiere decir que aunque la llamada a la función se generará igual que siempre (sabiendo que *mover_cursor()* es la número 4):

```
generar(_lla,funciones[4])
```

la función del código de cabecera se debe declarar recibiendo los parámetros en orden inverso, de este modo:

```
void mover_cursor(int y,int x){
  // Código...
}
```

Y como la función tiene dos parámetros (cuatro bytes), tras generar el LLA se hará:

```
generar(_asp,4)
```

para generar código que desapile los parámetros de la función.

## PARÁMETROS POR VARIABLE

Cuando la función interna requiere un parámetro por variable, como *leer_tecla()*, se complica un poco el paso de los parámetros a la función.

Supongamos que en el programa se invoca a la función anterior pasándole como parámetro la variable "n". Si en la función de análisis se invocara a *an_EXPRESION()* para analizar el parámetro, se generaría código para evaluar la expresión "n", es decir, se generaría un LOE que apilase el valor de la variable "n".

Pero la implementación de *leer_tecla()* en startup.cpp no requiere como parámetro el valor de "n", sino la dirección de "n", ya que se define del siguiente modo:

```
void leer_tecla(int*n){
  // Código...
}
```

Por ello no se puede utilizar *an_EXPRESION()*. En su lugar se utiliza el procedimiento de análisis *an_DATO()* que ya se utilizó en las sentencias de asignación. Éste analiza un dato (o una tabla con una expresión como índice) y se le puede indicar si debe o no debe generar código para apilarlo.

Se invocará a *an_DATO()* para analizar la variable "n" indicando que no genere código para apilar el valor de la misma. Como este procedimiento de análisis además devuelve el registro (de la tabla de objetos, se entiende) que corresponde a la variable "n", se puede leer en él la dirección de la misma.

**Cada compilador añade a los programas su propio código START UP, no tiene que ver sólo con el lenguaje de programación**

### CUADRO 1

Si lo único que se quiere hacer es cambiar el código de inicialización o finalización de Letra, simplemente se tiene que modificar la función correspondiente dentro del *start up*, volver a compilarlo, y copiar el ejecutable generado (startup.com) al directorio en el que reside el compilador.

Pero para cambiar o añadir nuevas funciones internas, se deben seguir los siguientes pasos (supóngase que se va añadir una nueva función interna llamada sonido(frecuencia,duración))

-Primero añadir un código a la lista que aparece en "lexico.h", en el punto marcado como "Nombres de las funciones especiales", que en este caso se denominará p_sonido.

-En segundo lugar, añadir la palabra reservada ("sonido") que va a identificar a la nueva función en el analizador léxico (en "lexico.cpp", en la misma posición relativa en la que se insertó el código en "lexico.h").

-Añadir un nuevo "case" en an_FUNCION_INTERNA() de "letra.cpp", para que se llame a la función de análisis cuando se encuentre la nueva palabra reservada.

-Crear la función de análisis correspondiente a la sentencia, teniendo en cuenta en este caso que la sintáxis para esta nueva sentencia será (también se debe crear un prototipo para esta nueva función):

```
<SONIDO> ::=
  p_sonido p_abrir_parentesis <expresión>
  p_coma <expresión> p_cerrar_paréntesis
```

-Añadir a la lista funciones[ ] del código de cabecera ("startup.cpp") un puntero a la nueva función.

-Por último implementar la función dentro de "startup.cpp", con el siguiente prototipo (se recuerda que los parámetros se reciben en orden inverso):

```
void sonido(int duracion, int frecuencia);
```

Compilar el compilador y el *start up*, poniendo ambos ejecutables en el mismo directorio. Para no perderse es aconsejable seguir el ejemplo de otra función interna similar, como en este caso podría ser mover_cursor(x,y).

**Cómo añadir una función interna.**

Para apilar el puntero a "n" que la función *leer_tecla* espera como parámetro, se generará el siguiente código (que apilará la dirección que se le haya asignado al dato "n"):

*generar(_loc,dirección_de_n)*

Después de apilar el parámetro se genera código para invocar a la función interna y desapilar el parámetro de la misma forma que siempre (con un LLA y un ASP).

Pero, ¿qué pasa cuando se pasa como parámetro una tabla, como en el siguiente ejemplo?

*leer_tecla(t[2+3])*

Para el análisis del parámetro se vuelve a utilizar an_DATO(), pero ahora éste generará código para evaluar el índice (2+3). La función *leer_tecla()* del *start up* espera que se le pase un puntero a la dirección en la que debe devolver el resultado (la dirección de t[5]), ésta dirección se puede calcular como: (supóngase que la dirección de "t", que se obtiene de su registro en la tabla de objetos, es por ejemplo 808)

*5*2+808*

Se recuerda que el índice se multiplica por 2 debido a que cada posición de la tabla ocupa 2 bytes. Para calcular dicha dirección se genera código de la siguiente forma (teniendo en cuenta que tras *an_DATO()* ya se ha generado código para apilar el índice):

*generar(_loc,2)*
*generar(_mul)*
*generar(_loc,808)*
*generar(_add)*

Se multiplica el índice por 2 y se suma la dirección de inicio de la tabla, una vez se ejecuten todas las instrucciones generadas hasta ahora:

*LOC 2; LOC 3; SUM; LOC 2; MUL; LOC 808; ADD;*

Se tendrá apilado el valor 818, que es la dirección de t[2+3], por lo que ya se puede generar la llamada a la función como de costumbre (LLA y ASP).

Para resolver el problema anterior, en lugar de generar esas cuatro instrucciones (loc 2, mul, loc 808, add) se podría haber añadido una nueva instrucción al lenguaje EML que recibiendo como parámetro la dirección de una tabla y el índice apilado, obtuviera directamente la dirección de la posición de la tabla. Esta nueva instrucción se podría denominar de cualquier forma, por ejemplo XYZ. A continuación se da la cadena de código que implementaría dicha instrucción en código 80x86 y las instrucciones ensamblador correspondientes:

***XYZ(4):*** *58 01 c0 05 00 00 50*
***XYZ:*** *pop ax; add ax,ax; add ax,0; push ax*

Que no se haya introducido aquí esta instrucción en el código EML no responde a ningún motivo especial, simplemente se pretende ilustrar cómo no todos los problemas se tienen que resolver inventando una nueva instrucción del código intermedio. Aun así, durante el diseño de un lenguaje, se suelen añadir todas las instrucciones de código intermedio que se van necesitando.

## LA FUNCIÓN ESCRIBIR

Con las explicaciones que se han dado se pueden resolver ya todas las llamadas a las funciones internas menos una: la función escribir().

Esta función es especial, ya que debe pasar como parámetro un literal en lugar de un dato o la dirección de un dato. La implementación de la función interna escribir() en el código de cabecera se define así:

```
void escribir(char*){
  // Código ...
}
```

**La comunicación entre el código de cabecera y el compilador de Letra se realiza utilizando variables de la BIOS**

Es decir, la función en C espera un puntero al comienzo del literal. El problema principal es decidir dónde se va a guardar el literal. El compilador de Letra lo resuelve almacenando el literal en el vector mem[ ] en el punto en el que se esté generando código.

Pero no se puede poner un literal en medio del código máquina directamente, pues un literal no es código ejecutable por el procesador, por lo que inmediatamente antes del literal se generará un salto para que no se ejecute el literal.

La longitud del literal se puede calcular, por lo que se puede saber cuánto espacio va a ocupar en memoria y por tanto, hasta dónde se tiene que hacer el salto. Para generar este salto se vuelve a utilizar la instrucción IR:

**CUADRO 2**

En el disco se encuentran dos códigos de cabecera equivalentes pero distintos: el que aquí se ha explicado, que está programado en C, y el otro en ensamblador.

Se entrega este último por ser más eficiente en cuanto a espacio (ocupa 483 bytes en lugar de los 4802 bytes que ocupa el programado en C), no obstante no se explicará detalladamente su funcionamiento por ser idéntico al otro.

Si se quiere modificar el código de cabecera por cualquier motivo (como variar el lenguaje o las funciones internas), se debe decidir cual de los dos se va a utilizar, pues como es lógico una vez se modifique uno (se le añada una nueva función), ya no se podrá utilizar el otro (que no tendrá dicha función).

Se aconseja, que sí no se tienen conocimientos de ensamblador se utilice el código de cabecera programado en C.

**Acerca del *start up* ensamblador.**

*generar(_ir,fin_literal)*

Entonces el compilador inserta el literal en mem[ ] y después generará (para apilar la dirección del literal) un LOC de este modo:

*generar(_loc,dirección_literal)*

Al ejecutarse se tendrá apilado el parámetro que requiere la función, por lo que se finalizará la generación de código para invocar a esta función con las instrucciones LLA para llamarla y ASP para desapilar el parámetro.

En el artículo anterior se explicó como se generaba un fichero de órdenes para que una vez generado el código máquina, DEBUG creara un listado con el desensamble del mismo (para poder ver el "código máquina" que se genera). Dentro de la función *an_ESCRIBIR()* se le tiene que añadir una orden para que no desensamble el literal, esto se hace como se explicó en dicho artículo.

En otros lenguajes de programación son las mismas funciones las que quitan los parámetros de la pila antes de retornar, en estos lenguajes no es necesaria la instrucción ASP tras cada llamada a una función con parámetros.

## Y GRABAR EL EJECUTABLE

El compilador se ha finalizado, sólo resta grabar el vector mem[ ] a disco como un ejecutable, pudiendo darse por finalizada la compilación.

Queda únicamente algún pequeño retoque, como puede ser el añadir al final de todos los programas una instrucción *RET*, por si al usuario se le olvida acabar el programa con una sentencia *VOLVER* (pues en ese caso se ejecutaría código más allá del generado, con resultados del todo imprevisibles).

Además el usuario espera que si se compila por ejemplo el programa "tablas.lt", se genere el código máquina en un fichero denominado "tablas.com" y no "código.x86" como se hacía hasta ahora.

El resto de modificaciones que se quieran hacer sobre el compilador son opcionales, por ejemplo, se puede eliminar la generación de listados, ya que estos son solo informativos.

## STARTUP.CPP

El código de cabecera de Letra es un programa COM que como se ha dicho se construye de manera que pueda ejecutarse de dos formas diferentes:

- Informativa: si se ejecuta directamente informará al compilador de la dirección en la que está la tabla de funciones y de la dirección final del código de cabecera.

- Completa: si se ejecuta con código añadido y con la dirección en la que debe comenzar a ejecutarse este código en *funciones[0]*, entonces se llamará a la función *inicializar()*, al programa compilado y por último a *finalizar()*.

Inicialmente se tiene en *funciones[0]* un puntero al propio main() del *start up;* si el compilador no modifica este valor, el código de cabecera se ejecutará de la primera forma, siendo el funcionamiento el siguiente:

Para comunicar al compilador las dos direcciones se utiliza como almacenamiento temporal dos variables de la BIOS que no estan siendo utilizadas para otra cosa (0x45C y 0x45E variables que teóricamente contienen la posición del cursor en las páginas de vídeo 7 y 8). Para obtener la dirección final del código de cabecera se realiza una operación sencilla, se pide 1 byte de memoria (*malloc(1)*) y se toma la dirección devuelta como la primera posición libre tras el código de cabecera (a partir de la misma el compilador de Letra generará el código máquina). Una vez puestas estas dos direcciones en las variables de la BIOS, el *start up* finalizará su ejecución.

Cuando en *funciones[0]* no hay un puntero a main(), el *start up* se ejecuta de la segunda forma. Primero se invoca a *inicializar()*, donde se realizan los siguientes pasos:

-Poner el modo de vídeo 3 (modo texto 80x25).
-Ocultar el cursor (estará invisible durante toda la ejecución del programa).
-Instalar la interrupción de reloj (que simplemente incrementa un contador).
-Instalar la interrupción de teclado (hace que se aborte la ejecución si se pulsa la tecla ESC, es decir, salta directamente a *finalizar()*).

- Reprogramar la interrupción de reloj para que se ejecute cada centésima.

Tras todos estos pasos se llama a la dirección que hay en *funciones[0]* (el compilador ha puesto aquí la dirección en la que comienza el código máquina). Cuando éste retorne (o bien se pulse ESC), se invocará a *finalizar()*, realizándose los siguientes pasos:

- Reprogramar la interrupción de reloj para que se ejecute nuevamente 18'2 veces por segundo.
- Desinstalar las interrupciones de reloj y teclado.

- Mostrar nuevamente el cursor.

- Borrar la pantalla.

Y entonces se retorna al sistema operativo. Dentro de "startup.cpp" están las funciones internas de Letra, estas son muy sencillas en su implementación, por lo que aquí no se explicarán en detalle (ver el listado en el disco que acompaña a la revista).

## PRÓXIMOS NÚMEROS

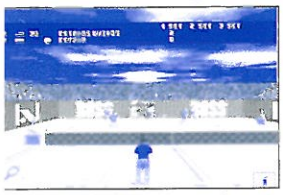
Espero que os haya parecido interesante esta serie de artículos, ostentosamente denominada "Construcción de un compilador", ya que en realidad se ha pasado muy por encima de muchos temas y se han saltado otros, buscando únicamente seguir un sencillo ejemplo práctico y sin entrar en complicadas teorías. Ruego perdonéis los errores cometidos en el camino y disculpéis las omisiones.

Se están preparando artículos más avanzados sobre compiladores, estos seguirán manteniendo el mismo nexo de unión que esta serie: el lenguaje Letra.

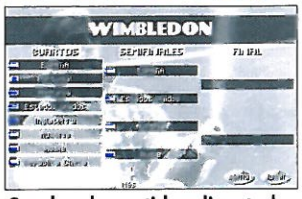
Gracias a aquellos que habéis seguido esta serie. Espero ver vuestros compiladores.

# VIRTUAL TENIS

## TODO EL TENIS MUNDIAL EN UN CD-ROM DE IMPACTO...



errenos de juego creados en D con nubes fractales.



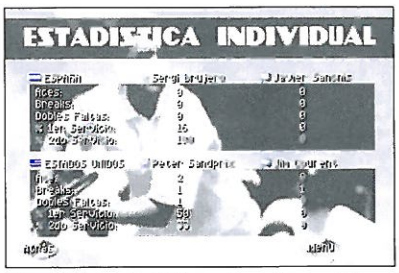

Cuadro de partidos disputados en el torneo en curso.

os torneos se pueden jugar n compañía de un amigo.

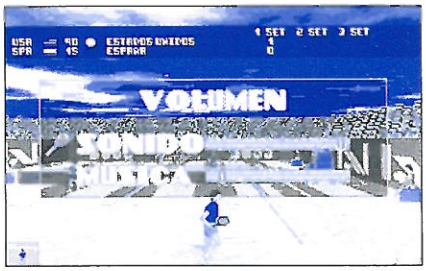
Control de los jugadores con teclado o Joystick.

osibilidad de jugar el Grand Slam completo: permite grabar orneos y temporadas a la mitad.

e pueden disputar los cuatro torneos del Grand Slam.




Completas estadísticas de cada partido.




Control independiente de música y sonido.

- Posibilidad de conectarse con distintos jugadores mediante red, módem y cable serie.
- Sistema de menúes totalmente intuitivo.
- Ayuda interactiva durante el desarrollo del juego.
- Fácil instalación en el disco duro.
- Jukebox para escuchar las distintas melodías que contiene.
- Cuatro diferentes superficies.
- Cada uno de los jugadores posee sus propias características.
- Permite elegir el número de sets que se desee disputar.
- Posibilidad de cambiar incluso el detalle gráfico.

CON LA GARANTÍA DE DDM DIGITAL DREAMS MULTIMEDIA

**Solicita VIRTUAL TENIS enviando este cupón o llamando al teléfono (91) 661.42.11* de 9 a 14 y de 15 a 18 , o por Fax: (91) 661.43.86**

Deseo que me envíen: ☐ **VIRTUAL TENIS por 2995 ptas. + 250 ptas. de gastos de envío.**

Nombre y apellidos.....Domicilio.....Población.....

Provincia.....C.P .....F. de nacimiento.....Profesión.....

**FORMA DE PAGO:**

Talón a ABETO EDITORIAL ☐ Contra-reembolso Teléfono ..... Firma ,

☐ Giro Postal nº.....de fecha.....

☐ Tarjeta de crédito VISA nº ☐☐☐☐ ☐☐☐☐ ☐☐☐☐ ☐☐☐☐

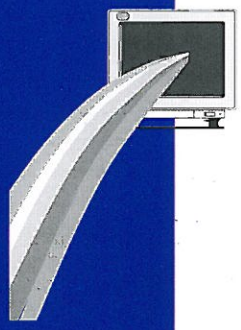
Fecha de caducidad de la tarjeta.....Nombre del titular, si es distinto.....

ABETO

Rellena este cupón y envíalo a:
ABETO EDITORIAL
C/ Aragoneses, 7.
28100 Pol. Ind. ALCOBENDAS (Madrid)

# EL CRTC CONTROLLER

*Santiago Romero, Miguel Cubas y Vicente Cubas*



Continuando con el importante tema de los registros VGA, que se dejará zanjado definitivamente al término de este artículo, se describirán a continuación los más importantes registros del Cathode Ray Tube Controller.

En las tablas 1 a 5 están descritos los registros a los que se tiene acceso mediante la programación del CRTC Controller y el significado de sus bits junto con los puertos índice en los que están situados.

La importancia de estos registros radica en su control sobre las condiciones en que aparece en el monitor la imagen contenida en memoria a partir del segmento VGA (A000h). A lo largo de las tablas de registros aparece varias veces el concepto de *scan-line counter, line counter* o, simplemente, *contador interno de líneas*. Este contador interno es una variable que contiene el número de scan-line por el que va el haz de electrones en su camino de refresco de la pantalla. Un *scan-line* es cada una de las líneas horizontales de las que se compone un gráfico, en este caso cada una de las líneas horizontales de la pantalla.

En los ejemplos incluidos en el CD de la revista aparecen listados pensados para ayudar a comprender el resultado que comportaría la modificación de algunos de estos registros, pero conviene que la base de esta experimentación la realice el lector, ya que mediante la librería VGAREGS.H y gracias a las tablas publicadas tiene total control sobre su tarjeta y debería ir investigando el significado de cada registro y el uso que podría dar de cada uno de ellos en sus programas.

Para ayudar en esta experimentación, y para comprender los valores que toman estos registros cuando tenemos iniciado un modo gráfico estándar, el autor incluye en el CD de la revista una sencilla utilidad (VGAREGS.EXE) que permitirá al lector explorar los registros de los modos estándar y *unchained* de la VGA.

## LA UTILIDAD VGAREGS

Esta utilidad está creada mediante la librería REGTABLE.H, que engloba funciones de carga de ficheros .UNC (*unchained*). Estos ficheros son un formato propio del programa Vgaregs que contienen simplemente una lista consecutiva de valores que hay que enviar a los registros para conseguir un determinado modo gráfico. Por ejemplo, se puede cargar en memoria el fichero 360x400.UNC mediante la función *LoadRegTable()*, y enviar estos valores a los puertos VGA gracias a la función *OutRegTable()*, para inicializar el modo unchained de 360x400 píxels. Esto se ha hecho para que los lectores de Solo Programadores dispongan de casi todos los modos X preparados para ser usados en los programas con las funciones implementadas en esta librería.

**La importancia de estos registros radica en su control sobre las condiciones de refresco**

**El CRTC Controller permite el control sobre el refresco de la pantalla y define cómo debe ser realizado éste, incluyendo funciones que posibilitan el desplazamiento de la pantalla por la video-memoria, base del scroll hardware. En el CD-ROM el lector encontrará un ejemplo de scroll vertical en modo x.**

| CRTC CONTROLLER, 3d4h | | |
|---|---|---|
| ÍNDICE: | NOMBRE: | DESCRIPCIÓN DEL REGISTRO: |
| 00h | Horizontal Total Register. (Read/Write) | Define el número total de carácteres en cada scan horizontal menos 5, incluyendo los carácteres correspondientes a espacios y borde.<br>Bits:<br>--------<br>0 - 7 (EGA) Número de carácteres horizontales menos 2.<br>(VGA) Número de carácteres horizontales menos 5. |
| 01h | Horizontal Display End Register. (Read/Write) | Define el número total de carácteres visualizados en cada scan horizontal menos 1.<br>Bits:<br>--------<br>0 - 7 Número de carácteres horizontales a visualizar menos 1. |
| 02h | Start Horizontal Blanking Register. (Read/Write) | Controla el carácter en el que empieza el blanqueo de carácteres.<br>Bits:<br>--------<br>0 - 7 Número en el que empieza el contaje del blanqueo (blanking start count). |
| 03h | Horizontal Blank End Register. (Read/Write) | Define el carácter en que termina el blanqueo horizontal.<br>Bits:<br>--------<br>0 - 4 "Horizontal Blanking" acaba cuando los últimos 6 bits del "contador de carácteres" es igual a este valor. El sexto bit se puede encontrar en el puerto 3d4h, index 5 bit 7. En la EGA sólo se usan estos 5 bits.<br>5 - 6 Número de carácteres a esperar tras cada alcance al Horizontal Total Register.<br>7 (VGA) Acceso a los registros de control del Retrazo Vertical si está activado. |

Tabla 1. Registros del CRTC Controller (I).

No obstante, existe una utilidad shareware técnicamente más completa que Vgaregs, y ésta es *Tweak*. Si el lector puede conseguirla, adelante. Es una herramienta casi profesional de Robert Schmidt.

## LA LIBRERÍA REGTABLE

Fundamentalmente consta de las siguientes funciones:

- *InitRegTable():* Inicializa la estructura RegTable con los valores iniciales que necesita.
- *LoadRegTable( char *filename )*: Carga un fichero .UNC dentro de la estructura RegTable en memoria, leyendo todos sus registros y cargándolos.
- *SaveRegTable( char *filename )*: Salva hacia un archivo el contenido actual de la tabla en memoria.
- *InRegTable()*;:Lee el valor de todos los registros que son importantes para el modo X en la estructura.

La utilidad VGAREGS gestiona (carga y salva) estos ficheros y es capaz de mostrar los valores de los registros VGA en el modo de vídeo que se le indique, además de ofrecer la posibilidad de modificar estos valores y generar un .UNC preparado para hacer pruebas con los nuevos registros establecidos.

También puede ser usado para estudiar el valor que toman los registros en cualquier modo estándar. Por ejemplo, se puede estudiar cómo está compuesto internamente el modo 13h cargándolo en Vgaregs mediante la tecla M (de modo) y escribiendo 19 como respuesta al modo (siempre en formato decimal 13h = 19d) que se desee cargar. A continuación aparecen en pantalla los valores que los registros toman por defecto en ese modo.

Esto es tan sólo un ejemplo de lo que puede hacerse con esta sencilla utilidad, además de la posibilidad de crear nuestros propios modos X.

| CRTC CONTROLLER, 3d4h | | |
|---|---|---|
| ÍNDICE: | NOMBRE: | DESCRIPCIÓN DEL REGISTRO: |
| 04h | Start Horizontal Retrace Register. (Read/Write) | Define el carácter en el que empieza el Retrazo Horizontal.<br>Bits: (los más importantes)<br>--------<br>0 - 7 El Retrazo Horizontal comienza cuando el contador de carácteres (character counter) alcanza este valor. |
| 05h | End Horizontal Retrace Register. (Read/Write) | Define el carácter en el que acaba el Retrazo Horizontal.<br>Bits:<br>--------<br>7 (VGA) Bit 5 del registro 3d4h index 3, bits 0-4.<br>6 Espera del retrazo horizontal.<br>5 Espera del retrazo horizontal.<br>0 - 4 Fin del retrazo horizontal. |
| 06h | Vertical Total Register. (Read/Write) | Indica el número total de Scan lines (líneas horizontales) menos 2, de esta manera nos da una idea (incluyendo bordes y espacios en blanco) sobre la resolución vertical.<br>Bits:<br>--------<br>0 - 7 Número en el que empieza el contaje del blanqueo (blanking start count). |
| 07h | Overflow Register. (Read/Write) | Sus bits se usan como extensión de algunos registros para indicarnos valores de 9 bits, usando éste registro como noveno bit.<br>Bits:<br>--------<br>7 Bit 9 del Registro Vertical Retrace Start (index 10h).<br>6 Bit 9 del Registro Vertical Display End (index 12h).<br>5 Bit 9 del Registro Vertical Total (index 6h).<br>4 Bit 8 del Registro Line Compare (index 18h).<br>3 Bit 8 del Registro Start Vertical Blanking (index 15h).<br>2 Bit 8 del Registro Vertical Retrace Start (index 10h).<br>1 Bit 8 del Registro Vertical Display End (index 12h).<br>0 Bit 8 del Registro Vertical Total (index 6h). |

Tabla 2. Registros del CRTC Controller (II).

| CRTC CONTROLLER, 3d4h | | |
|---|---|---|
| ÍNDICE : | NOMBRE: | DESCRIPCIÓN DEL REGISTRO: |
| 08h | Preset Row Scan Register. (Read/Write) | Indica que scan line de cada carácter será el primero en ser visualizado.<br>Bits: (los más importantes)<br>--------<br>0 - 4 Preset Row Scan |
| 09h | Max Scan Line Register / Character Height Register. (Read/Write) | En modo texto, los bits 0-5 indican la altura de los carácteres - 1.<br>Bits:<br>--------<br>7 Doble Scan.<br>6 bit 9 para el Line Compare Register (index 18h)<br>5 bit 9 para el Start Vertical Blank Register (index 15h).<br>0 - 4 Máxima línea de Scan. |
| 0Ah | Cursor Start Register. (Read/Write) | Indica el scan line en que comienza el cursor tipo bloque.<br>Bits:<br>--------<br>5 Si está a 1, desactiva el cursor.<br>0 - 4 Scan Line en que comienza el cursor. |
| 0Bh | Cursor End Register. (Read/Write) | Bits:<br>--------<br>0 - 4 Scan Line en que acaba en cursor. |
| 0Ch | Start Address Register. (Read/Write) | Byte ALTO de una dirección 16 bits que completa el registro 0Dh como byte bajo, y que indican el offset dentro del segmento de la VGA (0xA000) en que el haz de electrones debe leer bytes para redibujarlos en pantalla en la esquina superior izquierda. Ver texto del artículo. |
| 0Dh | Start Address Register. (R/W) | Byte Bajo del Start Address Register index 0ch. |

Tabla 3. Registros del CRTC Controller (III).

verticales como horizontales, y para el *enmascaramiento de planos*.

Enmascarar significa tapar, cubrir. Esto es lo que hace la función *MaskPlanes()* que permite prohibir el acceso a cualquiera de los 4 planos de la VGA. A esta función se le pasa como parámetro un byte cuyos últimos cuatro bits indican 1 (acceso permitido) o 0 (acceso no permitido), de manera que el bit 0 representa al plano 0, el bit 1 al plano 1, etc...

Si se quisiera permitir sólo la escritura a los planos 0 y 2, se le debería pasar como parámetro el valor 5 (0101b). Es decir, cada bit actúa como interruptor para un plano (el bit 0 para el plano 0, etc). Para permitir la escritura en los cuatro planos, bastaría con escribir *MaskPlanes( 0xF );* ( 0xf = 15 = 1111b ), tal y como hace la función *EnableAllPlanes()*.

El enmascaramiento de los planos se realiza mediante el Map Mask Register del VGA Sequencer, como

- *OutRegTable();*: Manda hacia los registros VGA los valores que contiene la tabla.

Si se quisiera cargar el fichero generado con Vgaregs 256x240.unc en un programa propio, por ejemplo, bastaría con que se encontrara en el mismo directorio que el .EXE generado desde un listado que incluyera las líneas:

```
InitRegTable();
LoadRegTable( "256x240.unc" );
OutRegTable();
```

Con esta librería se pretende proporcionar al lector un control sobre todos los modos X, aunque en los artículos se profundice sobre los 320 píxels horizontales de resolución.

## NUEVAS FUNCIONES

Además de las nuevas funciones *SetAddress()* y *SetPage()*, que serán comentadas en el próximo número, en la librería VGAREGS.H se han implementado funciones relacionadas con el control de los retrazos, tanto

| CRTC CONTROLLER, 3d4h | | |
|---|---|---|
| ÍNDICE : | NOMBRE: | DESCRIPCIÓN DEL REGISTRO: |
| 0Eh | Cursor Location Register. (R/W) | Es el byte alto del offset en memoria donde está situado el cursor. Junto con el registro 0Fh forma un offset de 16 bits. |
| 0Fh | Cursor Location Register. (R/W) | Byte bajo del offset de 16 bits donde se debe dibujar el cursor en la pantalla. |
| 10h | Vertical Retrace Start Register. (Read/Write) | 8 últimos bits del Vertical Retrace Start. El retrazo vertical comienza cuando el contador de líneas interno alcanza este valor. Es un valor de 10 bits. En este registro están los bits 0 a 7.<br>Bit 8: Se puede encontrar en 3d4h, index 7, bit 1<br>Bit 9: (VGA) Se puede encontrar en 3d4h, index 7 bit 7. |
| 11h | Vertical Retrace End Register. (Read/Write) | Bits:<br>--------<br>7 Protección contra escritura para los registros de index 0 a 7.<br>6 Si está a 1, selecciona 5 refrescos por scan en vez de 3.<br>5 Interrupciones verticales (IRQ2) desactivadas si está a 1.<br>4 Si está a 0 limpia las interrupciones verticales pendientes.<br>0 - 3 El retrazo vertical termina cuando los 4 últimos bits del contador interno de líneas alcanza este valor. |
| 12h | Vertical Display Enable End Register. (Read/Write) | Son los 8 bits menos significativos de un valor de 10 bits que la scan line menos 1 en que finaliza el refresco de la pantalla. (El refresco finaliza cuando el contador de líneas alcanza este valor).<br>Bit 8: Puede ser encontrado en 3d4h, index 7 bit 1.<br>Bit 9: Puede ser encontrado en 3d4h, index 7 bit 6. |
| 13h | Offset Register. (Read/Write) | (VGA) Número de bytes por scanline partido por 2, por 4 o por 6 dependiendo de si es un modo de bytes, word o dwords. |

Tabla 4. Registros del CRTC Controller (IV).

| CRTC CONTROLLER, 3d4h | | |
|---|---|---|
| ÍNDICE: | NOMBRE: | DESCRIPCIÓN DEL REGISTRO: |
| 14h | Underline Location Register. (Read/Write) | Subrayado para modos de texto monocromos.<br>Bits: (los más importantes)<br>--------<br>6 Si está a 1, modo de direccionamiento tipo DWORD.<br>5 Si está a 1, cambiar dirección de memoria cada 4 ciclos.<br>0 - 4 Lugar dentro del caracter para el subrayado. |
| 15h | Start Vertical Blanking Register. (Read/Write) | Los 8 LSB bits de 10 que indican en que scan-line comienza un espacio en blanco vertical (Vertical Blanking). Los bits restantes se encuentrar en los índices 7 y 9 del puerto 3d4h. |
| 16h | End Vertical Blanking Register. (Read/Write) | Los 8 LSB bits del valor que determina el Scan Line en que finaliza el Blanqueo vertical. |
| 17h | Mode Control Register. (Read/Write) | Bits:<br>--------<br>7 Permitir y/o activar retrazo vertical y horizontal si está a 1.<br>6 Si está a 1, modo de bytes. A 0, direccionamiento de words.<br>4 - 5 EGA y Word mode, no se usan en 13x.<br>3 Si es 1, incrementar el puntero de memoria sólo cada ciclo.<br>2 Si es 1, incrementar el "scanline counter" cada 2ª línea.<br>1 Compatibilidad con Hercules<br>0 Compatibilidad con CGA. |
| 18h | Line Compare Register. (Read/Write) | Contiene los 8 LSB bits de 10 que del Line Compare. Cuando el contador de líneas alcanza este valor, el Start Address Register (dirección de lo que vemos en pantalla) es reseteado a 0. Esto es útil para sistemas de Split Screen o pantalla partida.<br>Bit 8: Está en 3d4h, index 7 bit 4.<br>Bit 9: Está en 3d4h, index 9 bit 6. |

**Tabla 5. Registros del CRTC Controller (V).**

puede verse en las tablas de registros publicadas en el número anterior de la revista.

Un ejemplo de enmascaramiento de planos se ve en el listado del disco *ejemplo4.c*, donde se realiza un dibujo después de enmascarar el plano 0 que, como refleja el resultado, queda protegido contra escritura.

## RegTable.h gestiona ficheros .UNC preparados para la experimentación con la VGA

### DOCUMENTOS DE INFORMACIÓN

En el soporte de software que acompaña a la revista se incluye el subdirectorio DOCS, dentro del apartado de modos X, que incluye varios documentos de texto escritos por importantes programadores dentro de la escena que constituye el modo X.

De Michael Abrash se encuentran los listados *SETMODEX.ASM* y *SHOWPAGE.ASM*. El primero muestra la inicialización del modo X que diseñó el "padre" de los modos unchained. El segundo muestra la función llamable desde C que M. Abrash creó para cambiar el Start Address Register.

De Robert Schmidt está *MODEX.TXT*, un documento que resulta una excelente introducción al modo X, extraido del programa TWEAK.

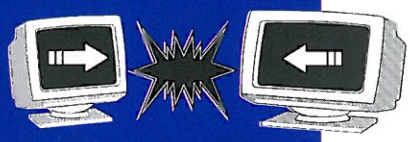
El documento *VGA.DOC*, de diversos autores, resulta imprescindible, pues acumula la definición de todos los registros VGA, incluyendo sus puertos, índices y el valor de sus bits. Una extracción de los registros que controla el CRTC se ha realizado hacia el documento *CRTC.DOC*.

### PRÓXIMA ENTREGA

En el próximo número se verá como inicializar la memoria VGA de los modos X para disponer de otras resoluciones de pantallas como 2x2 o 4x1, en vez de 4 pantallas verticales como en la figura 1, consiguiendo mediante el desplazamiento por la memoria realizar scrolles multidireccionales para demostraciones gráficas. Se dispondrá además de una versión mucho más potente de la librería de modo X, de manera que sea estándar para todas las resoluciones de páginas de 320x200 y 320x240. Hasta entonces, en los ejemplos del CD-ROM el lector puede encontrar un scroll vertical en modo X que puede ir estudiando como anticipo del próximo número.

¿SABE LO QUE SE PIERDE SI NO REGISTRA SUS APLICACIONES PARA MICROSOFT® WINDOWS 95?
✓ Soporte técnico gratuito por tiempo limitado.
✓ Suscripción gratuita a la revista de los Usuarios de productos Microsoft.
✓ Ofertas en actualizaciones, eventos, seminarios, cursos, etc.
✓ Tener la seguridad de que sus programas de software son legales.
Envíe ya su tarjeta de registro.
Para más información llámenos al telf.: (91) 804 00 96
Microsoft
¿HASTA DONDE QUIERES LLEGAR HOY?
Office Standard
PC MEDIA

SERIE COMUNICACIONES



# COMUNICACIÓN POR EL PUERTO SERIE

*Fernando de la Villa*

Escribir un programa de comunicaciones está lejos de ser una tarea trivial. La comunicación entre ordenadores es un tema muy amplio, que precisa de conocimientos teóricos sobre aspectos muy diversos y, sobre todo, de mucha práctica. Aquí nos ocuparemos solamente de una pequeña parcela de este complejo mundo que parece no tener límites.

El objetivo del artículo es desarrollar un programa de comunicaciones lo más sencillo posible capaz de transmitir un fichero de una máquina a otra a través del puerto serie. Para ello hay que establecer un soporte físico, programar el chip UART (Universal Asynchronous Receiver/Transmitter), escribir nuestra propia rutina de servicio de interrupción y definir un protocolo de comunicaciones, entre otras cosas.

**Es posible que el tema de la comunicación entre ordenadores sea uno de los más extensos y complejos del mundo de la informática. En el presente artículo se expone, desde el principio, la manera de transmitir ficheros entre dos ordenadores a través del puerto serie**

## CONCEPTOS BÁSICOS

Antes de empezar, es necesario dejar claros algunos de los conceptos básicos relacionados con las comunicaciones serie para comprender todos los pasos a realizar.

La comunicación serie consiste en enviar por una línea de comunicación una serie de datos en forma de bits, un bit detrás de otro (de ahí el nombre). El número de bits de datos enviados puede ser 5, 6, 7 u 8. Por razones de sincronismo y para detectar posibles errores, los bits de datos van acompañados de un bit de comienzo (con valor 0) transmitido en primer lugar y uno o más bits de parada (con valor 1), transmitidos después de los bits de datos. Opcionalmente puede utilizarse un bit denominado "de paridad" cuyo objetivo es la detección básica de errores. La paridad puede ser par o impar. En el primer caso, el bit de paridad complementa a los bits de datos para que en total el número de bits a 1 sea par. Si en recepción este bit no tiene el valor esperado, se supone que se ha producido un error.

Otro factor importante es la velocidad a la que se establece la comunicación. Se mide en baudios, que es la frecuencia de los impulsos eléctricos que viajan por la línea. Las velocidades más habituales son: 115.200, 57.600, 38.400, 28.800, 14.400, 9.600, 7.200, 4.800, 2.400 y 1.200 baudios.

## EL SOPORTE FÍSICO

Toda comunicación requiere de un soporte físico a través del cual se pueda realizar el intercambio de información. Pensando en nuestro problema concreto, una posible solución sería la utilización de dos modem conectados mediante un cable telefónico normal. Sin embargo, esta solución es cara y no permite alcanzar grandes velocidades salvo que se utilicen modem de banda base. Otra solución sería la instalación de una red de área local, pero tampoco es una opción barata y, por lo general, es muy costoso su montaje y configuración. La mejor solución en este caso es la utilización de un cable de los denominados "modem nulo", que permite la conexión directa entre los puertos serie de ambos ordenadores. El mayor problema de este tipo de conexión es la longitud del cable, que no puede ser mayor de 15 metros. Como contrapartida, es sin duda la solución más barata y la más fácil de instalar.

## EL INTERFAZ RS-232

Los puertos serie de los PC utilizan el interfaz RS-232 para comunicarse con el exterior, y hay que valerse de él para poder establecer la comunicación. El RS-232C es un interfaz entre un equipo de comunicación de datos (DCE) y un equipo terminal de datos (DTE), o lo que es lo mismo, un interfaz entre el módem y el ordenador. La intención es conectar directamente dos ordenadores sin la mediación de ningún módem, por lo que será necesario "engañar" al interfaz mediante un cableado especial conocido como módem nulo.

Los cables o adaptadores módem nulo pueden encontrarse fácilmente en establecimientos especializados y en tiendas de electrónica. Sin embargo, los más mañosos pueden construirse ellos mismos el dispositivo. Se necesita un cable de 3 hilos como mínimo y dos conectores hembra de 9 ó 25 pines (DB-9 y DB-25, respectivamente), según los conectores de los puertos serie de los ordenadores a conectar.

Para comprender mejor el truco del cable módem nulo, se detalla a continuación el significado de los pines más utilizados:

- TD (*Transmitted Data*): Es el pin de transmisión de datos. Sirve para enviar los datos desde el ordenador hacia el módem.
- RD (*Received Data*): Recepción de datos. A través de este pin se realiza la transmisión de datos desde el módem hacia el ordenador.
- RTS (*Request To Send*): Significa petición para enviar. Se utiliza para indicar al modem que el ordenador quiere enviarle datos. El ordenador debe esperar a que el modem le de permiso a través del siguiente pin.
- CTS (*Clear To Send*): Listo para el envío. Mediante este pin el modem informa al ordenador de que puede comenzar el envío. Se utiliza junto con RTS.
- GND (*Signal Common*): Este pin se utiliza como referencia para la correcta generación e interpretación de señales por parte de ambos equipos. Debe estar conectado siempre para poder realizar la comunicación sin problemas.
- DCD (*Data Carrier Detect*): Se trata del pin de detección de señal de línea en recepción. Sirve para indicar que el modem ha detectado una señal portadora de datos.

El cable módem nulo básico se construye conectando ambos pines GND, y el pin TD de un interfaz al RD del otro y viceversa. Sin embargo, es aconsejable la utilización de otros pines, como RTS y CTS de un interfaz conectados al DCD del otro (y viceversa). En las tablas 2 y 3 se detalla la numeración de los pines de ambos tipos de conector. En la figura 1 puede verse el cableado correspondiente al cable modem nulo básico y en la figura 2 se muestran los esquemas de cableado de las tres configuraciones posibles: los dos conectores de 9 pines, los dos de 25 pines y uno de 9 con otro de 25 pines.

## LA UART

El transmisor/receptor asíncrono universal es el corazón de la comunicación mediante los puertos serie. Se trata de un chip, generalmente el 8250, encargado de la gestión de todos los aspectos inherentes a la transmisión y recepción de datos mediante el interfaz RS-232. Se trata de un dispositivo programable capaz de gestionar paridad, bits de comienzo y parada, diferentes números de bits de datos y distintas velocidades de comunicación, que además incorpora un sistema de interrupciones por prioridades.

Este chip puede ser programado a través de sus diez registros. En un PC, cada puerto de comunicaciones tiene asociado una dirección base de memoria a partir de la cual dichos registros pueden ser accedidos. En la tabla 1 se detallan las direcciones base de cada puerto y en la tabla 4 se muestran las direcciones de los registros respecto a la dirección base. Más de la mitad de estos registros están formados por campos de bits, o lo que es lo mismo, cada uno de sus bits tiene un significado en concreto. Dichos registros son los siguientes:

- *Buffer* de transmisión (*TX Buffer*): Este es el registro en el que se escriben, uno a uno, los datos a enviar por el puerto serie. Si el formato de datos indica menos de 8 bits de información, sólo será enviado el número adecuado de bits, ignorando los bits sobrantes.
- *Buffer* de recepción (*RX Buffer*): El registro de recepción es el lugar donde la UART coloca, uno a uno, los datos recibidos en el puerto serie. En este registro siempre hay 8 bits, por lo que será necesario desechar los bits sobrantes mediante una máscara, en caso necesario. Cuando se recibe un nuevo dato, el anterior es sobrescrito. Por tanto, si los datos no son leídos con la suficiente rapidez se corre el riesgo de perder algunos de ellos. Más adelante se verá una forma de solucionar este problema.
- Byte menos significativo del divisor de frecuencia (*Divisor Latch LSB*): Mediante este registro y el siguiente se establece la velocidad en baudios a la que debe producirse la comunicación. En él se guarda el valor del byte bajo del divisor de frecuencia. En seguida se verá cómo se calcula este valor.
- Byte más significativo del divisor de frecuencia (*Divisor Latch MSB*): Este registro se complementa con el anterior. Su función es la de recibir el valor del byte alto del divisor de frecuencia.
- Habilitación de interrupciones (*Interrupt Enable Register*): Mediante el registro de habilitación de interrupciones se habilita la generación de los cuatro tipos de interrupciones diferentes que soporta el chip. El significado de los 8 bits que lo componen es el siguiente:

0: Si es igual a 1 se activa la interrupción por datos disponibles. Es decir, la interrupción saltará en el momento que un dato esté disponible para ser leído.

1: Si vale 1 se activa la interrupción por registro de transmisión vacío. La inte-

**TABLA 1**

| Puerto serie | Dirección | IRQ | Vector |
|---|---|---|---|
| COM1 | 3F8h | 4 | 12 |
| COM2 | 2F8h | 3 | 11 |
| COM3 | 3E8h | 4 | 12 |
| COM4 | 2E8h | 3 | 11 |

**Datos de configuración de los puertos serie en un PC.**

rrupción se producirá cuando la UART pueda recoger un nuevo dato para transmitir.

2: Si está activo habilita la interrupción de estado de línea. Hay cuatro causas que pueden lanzar esta interrupción: error de paridad, error de *overrun*, error de rango o por un *BREAK* en recepción. Las cuatro están reflejadas en diferentes bits del registro de estado de línea.

3: Cuando está activo se puede producir la interrupción de estado del módem. También son cuatro las posibles causas de la generación de esta interrupción: cambio del estado del CTS, DSR, RI o RLSD. Las cuatro están reflejadas en el registro de estado del módem.

4 al 7: Están siempre a 0.

● Identificación de interrupciones (*Interrupt Identification Register*): Su misión consiste en almacenar el número de identificación de la interrupción pendiente de mayor prioridad, si es que hay alguna. Cada una de las cuatro interrupciones posibles tiene una prioridad determinada. Siempre se atiende a la interrupción con mayor prioridad y mientras esté pendiente dicha interrupción no se atenderán otras. El significado de los 8 bits de este registro se especifica a continuación:

0: Es 0 si hay alguna interrupción pendiente, 1 si no la hay.

1 y 2: Número binario de 2 bits con el identificador de la interrupción. De mayor a menor prioridad, los valores y la interrupción asociada son:

3) Interrupción de estado de línea

2) Interrupción por datos disponibles

1) Interrupción por registro de transmisión vacío

0)Interrupción de estado del módem.

Cada una de estas interrupciones se da por atendida cuando se lee el registro con el que está asociada, excepto la interrupción por registro de transmisión vacío. En este último caso es preciso escribir en el registro de transmisión un nuevo dato o leer el registro de identificación de interrupción.

3 al 7: Siempre a 0.

● Control de línea (*Line Control Register*): Sirve para programar el formato de la comunicación y para seleccionar el acceso a algunos de los registros de la UART. Veamos el significado de sus bits:

**TABLA 2**

| Pin | Nombre | Función |
|---|---|---|
| 1 | No tiene | Masa |
| 2 | TD | Transmisión de datos |
| 3 | RD | Recepción de datos |
| 4 | RTS | Petición de envío |
| 5 | CTS | Listo para el envío |
| 6 | DSR | Equipo de datos preparado |
| 7 | GND | Retorno común (Masa de señales) |
| 8 | DCD | Detección de portadora de datos |
| 20 | DTR | Terminal de datos preparado |
| 22 | RI | Indicador de llamada |
| 23 | DSRD | Detección de velocidad de datos |

**Función de los pines en un conector DB-25.**

0 y 1: Especifican el número de bits de datos. Es un número binario de dos bits cuyo significado es el siguiente: 0 - 5 bits, 1 - 6 bits, 2 - 7 bits y 3 - 8 bits.

2: Número de bits de parada. Si es 0, se genera un sólo bit de parada. Si es uno, el número depende del valor de los bits 0 y 1 de este registro. Con 5 bits de datos se genera 1 bit y medio, y con el resto de valores se generan 2 bits.

3: Si es igual a 1, se genera el bit de paridad.

4: Si está activo se utiliza paridad impar. Si no, se usa paridad par. Para que este valor tenga efecto, el bit anterior deberá estar activo.

5: Es el bit "*stick parity*". Siempre que el bit 3 y éste bit estén activos, la paridad se transmite y se interpreta como un 0 lógico si el bit 4 está activo, o como un 1 lógico si está inactivo.

6: Cuando se activa produce un estado BREAK hasta que vuelva a desactivarse. El estado BREAK sirve para detener la comunicación y consiste en enviar todos los bits a 0, incluyendo el bit de parada.

7: Este es el *Divisor Latch Access Bit* (DLAB). Cuando vale 1 se accede a los dos registros del divisor de frecuencia en las direcciones base+0 y base+1. Si vale 0 se accede a los registros de transmisión y recepción en la dirección base+0 y al registro de identificación de interrupción en la dirección base+1. En la tabla 4 se ve la manera correcta de acceder a los registros.

● Control de modem (*Modem Control Register*): Se trata de un registro de 8 bits encargado de controlar el interfaz con el módem. Este es el contenido de sus bits:

0: Terminal de datos preparado (DTR). Si se activa, se fuerza que la señal DTR sea un 0 lógico, y viceversa.

1: Petición de envío (RTS). El comportamiento es similar al del bit anterior, pero con la señal RTS.

2: Señal auxiliar 1 (OUT1). Actúa de forma similar a los bits anteriores.

3: Señal auxiliar 2 (OUT2). Esta bit es de vital importancia para las interrupciones de la UART en un PC. Si el bit no está activo no se reconocerá ninguna de las interrupciones provocadas por la UART.

## La UART es el corazón de la comunicación mediante puertos serie

4: Loop. Sirve para hacer tests.

5 al 7: Siempre a 0.

● Estado de línea (*Line Status Register*): Es un registro de 8 bits que proporciona información sobre la transferencia de los datos. Los bits que lo componen son los siguientes:

0: Está activo cuando se ha recibido un nuevo dato.

1: Si está a 1 indica error de *overrun*. Significa que un nuevo dato recibido ha pisado el anterior, que no fue leído. Indica por tanto una pérdida de datos.

2: Se activa cuando se produce un error de paridad.

3: Cuando está activo indica un error de rango. Es decir, se ha recibido un dato con un bit de parada incorrecto.

4: Se activa en el momento que se detecta un estado de *BREAK*.

5: Está a 1 cuando el registro de transmisión está vacío y es seguro colocar un nuevo dato en dicho registro para realizar su transmisión.

6: Si vale 1 indica que el registro de transmisión interno de la UART está vacío. Este segundo registro de transmisión no es accesible directamente por el programador y la UART lo utiliza para poder recibir un nuevo dato en el registro de transmisión mientras está enviando el dato actual por la línea.

7: Siempre a 0.

● Estado de módem (*Modem Status Register*): El último de los registros de la UART refleja el estado de algunos de los pines del interfaz RS-232. Sus contenidos no necesitan más explicaciones. Son los siguientes:

0: *Delta Clear To Send* (DCTS).
1: *Delta Data Set Ready* (DSR).
2: *Trailing Edge Ring Indicator* (TERI).
3: *Delta Rx Line Signal Detect* (DRLSD).
4: *Clear To Send* (CTS).
5: *Data Set Ready* (DSR).
6: *Ring Indicator* (RI).
7: *Receive Line Signal Detect* (RLSD).

Uno de los detalles que ha quedado en el tintero es el cálculo del divisor de frecuencia para programar la velocidad de comunicaciones. La UART dispone de un generador programable de velocidad en baudios, capaz de leer un reloj de 1.8432 Mhz y partirlo por un divisor. La frecuencia de salida del generador es la velocidad en baudios multiplicado por 16. Según esto, la fórmula para calcular el divisor de frecuencia es:

*divisor = 1843200 / (velocidad * 16)*

El divisor obtenido se trata como un número de 16 bits, cuyos bytes de menor y mayor peso se cargan en los respectivos registros de la UART para obtener la velocidad de comunicación deseada.

## LAS INTERRUPCIONES

Una vez conocida la forma de programar el elemento más complicado, la UART, se puede pasar a examinar la forma de no perder esos valiosos datos que van a llegar por el puerto serie.

Para recoger los datos que van llegando a través del puerto de comunicaciones lo lógico sería preguntar continuamente a la UART por la llegada de un nuevo dato. Este método resulta inadecuado puesto que el programa no podría hacer otras operaciones mientras tanto. Por otro lado, interrogar de vez en cuando a la UART normalmente no será suficiente y se producirá la tan temida pérdida de datos.

**Existe el riesgo de perder datos si éstos no son leídos con la suficiente rapidez**

Como se ha visto anteriormente, una de las características más destacadas de este chip es la capacidad de generar una interrupción ante la aparición de determinadas condiciones. Una de estas condiciones es la llegada de un dato al puerto de comunicaciones, con lo que el problema queda solucionado. El programa puede realizar sus operaciones, y en el momento que llegue un dato, será interrumpido para recoger dicho dato y continuar más tarde con lo que estuviese haciendo.

La manera de utilizar las interrupciones de la UART es la siguiente:

1.- Escribir una rutina de servicio a la interrupción. Este es el código que responderá a la interrupción y que se encargará de almacenar el dato recibido en una memoria intermedia (buffer) hasta que sea procesado. Dicho buffer debe tener un tamaño lo suficientemente grande para almacenar todos los datos recibidos, mientras el programa no tiene tiempo de procesarlos.

2.- Guardar la dirección de la rutina anterior para restablecerla al finalizar el programa. Para ello hay que conocer el índice de la tabla de vectores de interrupción del sistema (las direcciones de las rutinas de tratamiento) correspondiente al puerto de comunicaciones que va a ser utilizado.

3.- Sustituir la dirección de la rutina anterior por la dirección de nuestra rutina.

4.- Programar los registros de la UART para el reconocimiento de la interrupción. Se hace en dos pasos: habilitar la interrupción por datos disponibles y activar la señal OUT2 mediante el registro de control del modem.

5.- Programar el PIC (*Programmable Interrupt Controler*) del PC para que informe de la interrupción a la CPU por la línea de interrupción (IRQ) correspondiente al puerto de comunicaciones. Esto se consigue enviando una máscara correspondiente a la IRQ del puerto de comunicaciones al registro de máscaras del PIC, y enviando el comando de fin de interrupción (EOI) al registro 2 de control de operación del PIC para limpiar las interrupciones pendientes.

6.- Limpiar las interrupciones pendientes para evitar posibles problemas. Para ello, se leen los registros de la UART correspondientes.

Los vectores de interrupción y las IRQ asociadas a los puertos pueden verse en la tabla 1.

## LA DETECCIÓN DE ERRORES

Cada vez el objetivo está más cerca. Sin embargo, no se había tenido en cuenta hasta ahora la posibilidad de que uno o varios datos recibidos no sean los correctos debido a alguna interferencia o perturbación que pueda afectar a la línea. La única forma de asegurar que la información recibida es exactamente la misma que la enviada es mediante la utilización de un método de detección de errores. Hay muchos, pero el más fiable y también el más utilizado es el CRC (Código de Redundancia Cíclica).

**TABLA 3**

| Pin | Nombre | Función |
|---|---|---|
| 1 | DCD | Detección de portadora de datos |
| 2 | RD | Recepción de datos |
| 3 | TD | Transmisión de datos |
| 4 | DTR | Terminal de datos preparado |
| 5 | GND | Retorno común (Masa de señales) |
| 6 | DSR | Equipo de datos preparado |
| 7 | RTS | Petición de envío |
| 8 | CTS | Listo para el envío |
| 9 | RI | Indicador de llamada |

**Función de los pines en un conector DB-9.**

El método CRC se basa en la matemática de polinomios. La explicación detallada del método matemático se sale de los objetivos de este artículo, por lo que no se entrará en detalles. Por el momento bastará con saber que el método se basa en considerar a los bits de los datos como polinomios y dividirlos entre un valor concreto. Lo realmente importante es el resto obtenido de la división.

## DISEÑO DEL PROTOCOLO DE COMUNICACIONES

Ahora ya se conoce la manera de enviar datos por la línea, pero esto no es suficiente. El objetivo del programa es realizar la transferencia de un fichero, por lo que habrá que especificarle al emisor cómo lo debe enviar y al receptor cómo debe interpretar los datos recibidos para reconstruir el fichero. El protocolo de comunicaciones es el que define la manera de entenderse entre los ordenadores.

Existen muchos protocolos de comunicaciones (*XModem*, *YModem*, *ZModem*, *BiModem*, etc.), pero lo mejor en este caso es optar por la creación de un protocolo de comunicaciones propio.

Nuestro protocolo está formado por dos tipos de tramas (conjuntos de información) bien diferenciadas: la trama cabecera y la trama de datos. La primera llevará la información referente al fichero como el nombre y la longitud. La segunda lleva los contenidos del fichero. Veámoslas con más detalle:

1.- Trama cabecera: está compuesta por los siguientes campos de información (por este orden):

- SOH: es un byte con el carácter ASCII *Start Of Header* (Comienzo de cabecera). Se utiliza para indicar al ordenador remoto que se comienza la transmisión de un fichero.
- Longitud del nombre del fichero: es un dato entero que indica la longitud del siguiente campo.
- Nombre del fichero: es una cadena ASCIIZ (acabada con un byte 0) con el nombre del fichero.
- Longitud del fichero: es un entero largo con la longitud total del fichero.
- Fecha y hora: es una estructura con los datos relativos a la fecha y hora de creación del fichero.
- CRC: es el código de redundancia cíclica (de 16 bits) de los datos anteriores. La validez de la trama recibida se comprueba calculando el valor del CRC de la trama completa (incluyendo su CRC). Si el valor obtenido es 0, la trama es correcta. En caso contrario, es errónea.

2.- Trama de datos: es más simple que la anterior. Se compone, por este orden, de los siguientes campos:

- Datos: 1024 bytes de datos del fichero. Si es la última trama de datos puede tener una longitud menor, para ajustarse exactamente a la longitud del fichero. Esto ocurrirá cuando la longitud del fichero no sea múltiplo de 1024.
- CRC: Código de Redundancia Cíclica de 16 bits. Se utiliza de la misma forma que en la trama de cabecera.

**TABLA 4**

| Dirección | Registro | Estado DLAB |
|---|---|---|
| Base + 0 | Buffer de transmisión | 0 |
| Base + 0 | Buffer de recepción | 0 |
| Base + 0 | Byte más significativo del divisor | 1 |
| Base + 1 | Byte menos significativo del divisor | 1 |
| Base + 1 | Habilitación de interrupciones | |
| Base + 2 | Identificación de interrupciones | |
| Base + 3 | Control de línea | |
| Base + 4 | Control del modem | |
| Base + 5 | Estado de línea | |
| Base + 6 | Estado del modem | |

**Direcciones de entrada/salida de los registros de la UART respecto a la dirección del puerto de comunicaciones**

Cada vez que recibe una trama, el receptor debe enviar un dato de 8 bits con el código ASCII ACK (*Acknowledge*), indicando que la trama es correcta. Si no lo es, enviará el código ASCII NAK (*Negative Acknowledge*) para pedir el reenvío de la trama. Si tras algunos intentos no se logra enviar bien una misma trama, se interrumpe la comunicación.


**BIBLIOGRAFÍA**

"Comunicaciones serie. Guía de referencia del programador en C" Joe Campbell. Editorial: Anaya Multimedia.

"Lenguaje ensamblador para microcomputadoras IBM. Para principiantes y avanzados" J. Terry Godfrey. Editorial: Prentice Hall.

"PC Interno" Michael Tischer. Editorial: Data Becker


## UTILIZACIÓN DEL PROGRAMA

El programa admite el paso de dos parámetros por la línea de comandos. El primero es el número de puerto de comunicaciones a utilizar y el segundo es el nombre del fichero a transmitir. Si sólo se especifica el puerto se asume la tarea de recepción, quedando el ordenador a la espera de recibir el código SOH. Si se especifican los dos parámetros, el ordenador comenzará a enviar la trama de cabecera, y si todo va bien continuará con las de datos. Para utilizar el programa es imprescindible arrancar primero en un ordenador el programa en modo recepción y posteriormente arrancarlo en el otro ordenador en modo envío.

Durante la comunicación el programa muestra mensajes por pantalla, mostrando el proceso. La comunicación se produce a 2400 baudios, con 8 bits de datos, sin paridad y con un bit de parada.

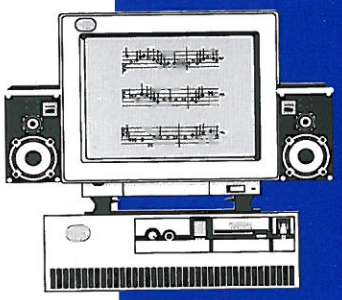
Por su parte, el código fuente del programa está lo suficientemente comentado como para ser comprendido sin demasiados problemas. También, por razones de claridad, no se han contemplado todos los posibles errores (disco lleno, etc.) con el fin de que el lector aprenda los fundamentos de la técnica y pueda "jugar" con el programa sin perderse en demasiados detalles que oscurecerían un programa que pretende ser más didáctico que otra cosa.

## CONSIDERACIONES FINALES

Ha quedado patente la complejidad y extensión del tema tratado en este artículo. Se ha intentado condensar, modestamente, toda la información relacionada en tan sólo unas pocas páginas. Esperamos que sea más que suficiente para que los lectores se animen a conocer mejor un mundo tan apasionante como es el de las comunicaciones por ordenador.

# FICHEROS WAV

*Agustín Guillén*



El formato WAV fue creado por IBM y Microsoft y fue introducido en el mercado de la mano de Windows 3.1. Sus siglas, RIFF WAVE, significan algo así como Formato de Ficheros para Intercambio de Recursos de Audio del tipo Waveform (Resource Interchange File Format Waveform Audio Format) y definen a la perfección sus intenciones.

Los ficheros WAV suelen ser bastante extensos, pues para poder albergar un minuto de sonidos en estéreo a una velocidad de muestreo de 44.1 Khz., con una resolución de 16 bits (calidad directamente extraible de un disco compacto), son necesarios unos 10 Mbytes de disco duro:

44.1 KiloHertzios
x
1000 Hertzios
x
60 segundos
x
2 bytes (16 bits)
x
2 canales (estéreo)
=
**10.584.000 bytes**

teniendo que sumarle todavía algunos bytes más para la cabecera y los datos auxiliares.

En el presente artículo, aparecerá con frecuencia el término *sample*, pues en el mundo del sonido digital tiene un par de acepciones, y ambas utilizadas frecuentemente : puede referirse a un único valor de salida dirigido al puerto del DAC (Conversor Digital a Analógico), en forma de un número de uno o o dos bytes ; o a un completo sonido independiente (formado por un gran número de los primeros *samples*). Ésto no debería preocupar a nadie, pues suele estar claro el tipo de *sample* del que se habla.

Todos los ficheros WAVE se atienen a la especificación RIFF (Rich Information File Format), que es todo un estándar en el intercambio de información multimedia.

## FORMATO

Los ficheros WAVE están divididos en bloques o *chunks*. Los distintos tipos de bloques que se pueden encontrar son:

Bloque de Formato
Bloque de Información
Bloque de Posiciones
Bloque de Lista de Reproducción
Bloque de Datos Asociados
Bloque de Datos

aunque cualquier programa reproductor de ficheros WAVE debe ser capaz de ignorar cualquier bloque que encuentre y del que desconozca su tratamiento. Las únicas reglas relativas a los bloques son:

- El fichero debe contener al menos un Bloque de Formato y un Bloque de Datos.
- El Bloque de Formato debe aparecer antes del Bloque de Datos.

## BLOQUE DE FORMATO

Especifica el formato del siguiente Bloque de Datos y se compone de dos partes bien diferenciadas:

*Datos Comunes*

Contiene los datos estándares para cualquier Bloque de Datos, sea cual sea

**En la anterior entrega estudiamos los ficheros voc, este mes se continúa la serie sobre formatos de ficheros de sonido con el formato para sonidos digitalizados más extendido en la actualidad: el formato RIFF WAVE, usualmente conocido como WAV.**

su tipo. Está formado por la estructura que se ve en la tabla 1:

**TABLA 1**

| Tamaño | Nombre | Descripción |
|---|---|---|
| WORD | wFormatTag | Categoría del Formato |
| WORD | wChannels | Número de Canales |
| DWORD | dwSamplesPerSec | Velocidad de Muestreo |
| DWORD | dwAvgBytesPerSec | Bytes por Segundo (para poder estimar un buffer) |
| WORD | wBlockAlign | Tamaño Mínimo del Bloque de Datos |

**Datos comunes.**

Mediante la Categoría del Formato (también se le podría denominar subformato), se indica el tipo exacto de los datos que contiene el fichero, además también define el contenido del subbloque Datos Específicos. Las Categorías definidas son las siguientes, aunque la creación de nuevos formatos está totalmente permitida, siguiendo unos sencillos requisitos. (Ver tabla 2)

**TABLA 2**

| Valor (Hex) | Descripción |
|---|---|
| 0001 | Microsoft Pulse Code Modulation (PCM) |
| 0101 | IBM mu-law |
| 0102 | IBM a-law |
| 0103 | IBM AVC Adaptative Differential/Delta Pulse Code Modulation |

**Categoría del formato.**

Más adelante se explicará el formato PCM, que es el más extendido y sin ningún tipo de compresión; para el resto, basta con una pequeña reseña: las Categorías *mu-law* y *a-law* tienen una codificación logarítmica de 8 bits, pero como se trata de una especie de pequeño número de punto flotante, abarcan un rango de 12 bits. La Categoría *a-law* es utilizada en Europa como un estándar para telefonía. Sobre ADPCM existen también varias normas, pero están orientadas en la compresión de voz (CCITT G.721 = ADPCM a 32 kbits/sec y G.723 = ADPCM a 24 y 40 kbits/sec).

*Datos Específicos*

Depende totalmente de la Categoría indicada en los Datos Comunes, por lo que su composición se explicará para cada Categoría.

## BLOQUE DE INFORMACIÓN

Aunque su uso actual está muy restringido, está diseñado para almacenar información importante sobre el fichero WAV y puede ser ampliado en el futuro. Suele contener 4 bytes indicando el número total de *samples*. Nótese que en los ficheros PCM no es imprescindible.

## BLOQUE DE POSICIONES

Sirve para identificar un número de posiciones o lugares dentro del fichero WAV. Se compone de 4 bytes iniciales conteniendo el contador de posiciones almacenadas y a continuación se encuentra una lista con las posiciones, cada una de ellas conteniendo la información especificada en la tabla 3.

**TABLA 3**

| Nombre | Descripción |
|---|---|
| dwName | Nombre de la Posición |
| dwPosition | Número del sample de la Posición, en orden de reproducción |
| fccChunk | Identificador del bloque que contiene la Posición |
| dwChunkStart | Offset en el fichero del inicio del *chunk* que contiene la Posición |
| dwBlockStart | Offset en el fichero del inicio del bloque que contiene la Posición |
| dwSampleOfffset | Offset del sample de la Posición, relativa al inicio del bloque |

**Bloque de Posiciones.**

## BLOQUE DE LISTA DE REPRODUCCIÓN

Indica un orden de reproducción para los puntos indicados mediante el Bloque de Posiciones. Los 4 bytes iniciales indican el número de puntos indicados y a continuación se listan dichos puntos, con el formato que se indica en la tabla 4:

**TABLA 4**

| Nombre | Descripción |
|---|---|
| dwName | Nombre de la Posición a reproducir |
| dwLength | Longitud de la reproducción, en *samples* |
| dwLoops | Número de veces a reproducir la sección definida |

**Bloque de Lista de Reproducción.**

## BLOQUE DE DATOS ASOCIADOS

Mediante esta sección, se pueden asignar nombres o etiquetas al fichero de sonido. Dispone de apartados para datos muy variados, como por ejemplo:

Nombre o título de un sonido
Texto explicativo o comentarios
Propósito del texto explicativo
Código del país al que va dirigido el texto
Código del lenguaje y del dialecto del texto
Código de página del texto

## BLOQUE DE DATOS

Depende totalmente de la Categoría del formato del fichero y como tan solo la primera Categoría no está registrada (PCM) y es de libre uso, y es la que habitualmente se encuentra en casi todos los ficheros WAVE, es la única que se va a describir en profundidad:

La Categoría PCM contendrá, como Datos Específicos, una WORD indicadora de los bits necesarios para cada *Sample* de cada canal. Los bits por *sample* (wBitsPerSample) deben ser iguales para cada canal que contenga el fichero, por lo que el dato del Bloque de Formato, Bytes por segundo (*dwAvgBytesPerSec*), coincidirá con la siguiente fórmula:

Número de Canales * Velocidad de Muestreo * Bits por *sample* / 8

y el dato Tamaño Mínimo del Bloque de Datos :

Número de Canales * Bits por *sample* / 8

El método de almacenamiento de los ficheros WAV pertenecientes a la Categoría PCM es el siguiente :

Cada *sample* o dato digital de sonido está contenido en un número entero y dicho número entero tendrá el mínimo tamaño necesario (en bytes) para albergar el tamaño de *sample* especificado (en la sección Datos Específicos). En la tabla 5 se aprecia con más claridad.

**TABLA 5**

| Tamaño del *sample* | Dato utilizado | Valor Mínimo | Valor Máximo |
|---|---|---|---|
| De 1 a 8 bits | Unsigned integer | 0 | 255 |
| Más de 9 bits | Signed integer | Valor máximo positivo de un Signed integer | Valor mínimo negativo de un Signed integer |

**Método de almacenamiento de ficheros WAV.**

Como herencia de la plataforma/microprocesador en donde se creó (Windows/Intel), el byte menos significativo estará almacenado primero; y los bits que no sean necesarios serán rellenados con ceros binarios y quedarán en los bits menos significativos. Con un ejemplo se verá más claro : para un *sample* de 10 bits, serán necesarios 2 bytes (16 bits disponibles) y los 6 bits sobrantes quedarán en la parte menos significativa (bit 5 a bit 0), del byte menos significativo (byte primero).

Los distintos *samples* almacenados en un fichero son almacenados bajo las reglas de la tabla 6:

**TABLA 6**

| | | | | | | |
|---|---|---|---|---|---|---|
| Sample 1 Canal 0 | Sample 2 Canal 0 | | | | | |
| **Para sonidos de 8 bits (1 byte) de varios canales (sonidos estéreo o superiores)** | | | | | | |
| Sample 1 Canal 0 | Sample 2 Canal 1 | Canal *n* | Canal 0 | Canal 1 | Canal *n* | |
| **Para sonidos de 16 bits (2 bytes) de varios canales (sonidos estéreo o superiores)** | | | | | | |
| Sample 1 Canal 0 byte bajo | Sample 2 Canal 1 byte alto | Canal *n* byte bajo | Canal 0 byte alto | Canal 1 byte bajo | Canal *n* byte alto | byte |
| bajo | byte alto | byte bajo | byte alto | byte bajo | byte alto | |

**Para sonidos de 8 bits (1 byte) de un solo canal (sonidos mono).**

## EJEMPLO

La interpretación de un fichero WAVE no representa grandes problemas. Lo mejor es conseguir, de cualquier compilador que genere código para Windows, las estructuras de los distintos bloques de datos que lo componen. Estas estructuras, creadas en C o en Pascal, se encuentran en las librerías o *includes* orientadas a multimedia.

En el disco que acompaña a la revista, se ha incluido un sencillo ejemplo para identificar e interpretar un fichero WAVE. No se ha intentado realizar una aplicación que lea cualquier fichero .WAV, tan sólo se trata de mostrar su estructuración. Para reproducirlo, simplemente restaría enviar al puerto de la tarjeta de sonido los datos hallados en el Bloque de Datos, previamente acotado. Ésto resulta muy sencillo con los ficheros pertenecientes a la Categoría PCM (la mayoría), pues para las otras Categorías se requiere una descompresión previa de los datos.

En el ejemplo se identifica previamente el fichero, para tratar de averiguar si es un fichero perteneciente a la norma RIFF ; a continuación se identifica si se trata de un fichero WAVE (hay que recordar que el estándar RIFF es utilizado para bastantes aplicaciones multimedia). Una vez que se tiene la certeza de que se trata de un fichero WAVE, se busca el Bloque de Formato, para informarse de la Categoría de Formato a la que pertenece. Tan sólo se interpretan los ficheros de la Categoría PCM, por lo que se genera un aviso, cuando no es así. Dentro del Bloque de Formato se continua leyendo el resto de la información contenida. Todo la estructura de los ficheros WAVE está orientada a bloques, por lo que resulta muy sencillo el continuar leyendo el siguiente. Como se espera que sea el Bloque de Datos, se genera un error cuando esto no sea así, aunque en una aplicación real, se deben ir saltando los bloques que no se desee interpretar, hasta llegar a los requeridos. Una vez que se ha localizado el Bloque de Datos, tan sólo se buscan en él los valores mínimo y máximo, por pura curiosidad, y se informa de ello.


**BUSCAMOS A LOS MEJORES**

- Programadores en C/C++
- Grafistas, diseñadores, dibujantes
- Expertos en lenguaje ensamblador
- Programadores 3D
- Expertos en Sonido
- Músicos con nociones de MOD, MIDI
- Programadores de juegos
- Animadores gráficos
- Infografistas con experiencia en 3D Studio, Photoshop, Deluxe Paint Animation
- Programadores con dominio de lenguajes multimedia: Authorware, Toolbook, Visual Basic, Macromedia Director, etc
- Expertos en comunicaciones

**PARA DESARROLLAR**

**SOFTWARE MULTIMEDIA:**

Presentaciones, libros interactivos, programas educativos, sistemas de comunicaciones, centros servidores de datos, videojuegos y mucho más...

SI ERES programador, músico o infografista y tienes ideas o proyectos en estudio de desarrollo ponte ya en contacto con nosotros.

TE OFRECEMOS las mejores condiciones y apoyo para producir tus programas y venderlos en el mercado nacional y extranjero. Formación en nuevas tecnologías y la mejor biblioteca de rutinas gráficas y sonido para desarrolladores.

Aportamos gráficos, rutinas o música, según las necesidades, para complementar cada proyecto.

Si estás interesado en unirte a una de las empresas más punteras en alta tecnología y desarrollo de software, no dejes pasar esta oportunidad. Envíanos carta con tus datos personales (curriculum vitae, con una muestra de tus anteriores trabajos) y un teléfono de contacto a:

DDM DIGITAL DREAMS MULTIMEDIA

*DIGITAL DREAMS MULTIMEDIA*
Ref. Programadores
C/ Vicente Muzas 15, 1º D - 28043 MADRID
Tf.: (91) 519.23.53 Fax (91) 413.55.77
BBS: (91) 519.75.75 Internet: ddm@servicom.es

# EL RAMDISK DE LINUX

*David Aparicio*

**El mes pasado se estudió el uso de un disco de arranque mediante un disco RAM. Ahora se profundizará en su implementación en el núcleo de Linux explicando el código que le hace funcionar.**

Siguiendo el plan que se comentaba hace un par de artículos, en esta entrega se va a explorar el núcleo de Linux, explicando el código que hace funcionar el disco RAM que se empleó en el disco de emergencia del pasado mes. Se trata de ver un aspecto teórico del sistema, acompañado de un uso práctico del mismo, para hacer más amena la serie.

El *ramdisk* es un dispositivo de bloque muy sencillo, que se resuelve de forma simple en el núcleo 1.2.13, el cual se sigue analizando. Hay que comentar que en las versiones 1.3.xx este código ha sido reorganizado por completo, ofreciendo nueva funcionalidad. Se mostrará también una comparación con los aspectos que diferencian a ambas implementaciones, pero centrándonos en la primera.

## FUNCIONALIDAD

El disco RAM de Linux está pensado, en la versión 1.2, para dar un apoyo importante al sistema cuando el directorio raíz se carga a partir de un disco flexible. Para evitar tener bloqueada la unidad de discos, así como para evitar corromper el disquete cuando se opera sobre él si se permiten escrituras (en un disco de arranque no suele caber un buen sistema de *shutdown*, y el apagado podría no ser muy fiable), y para acelerar el acceso al mismo, Linux copia dicha unidad sobre RAM, y ejecuta desde allí.

Este método sólo es inadecuado cuando se dispone de poca memoria, puesto que se ocupan 1,4 Mbytes que podrían ser muy necesarios para un sistema pequeño.

Se puede controlar cuando se desee un disco RAM mediante dos parámetros que deben utilizarse simultáneamente. Por un lado, la unidad marcada como raíz del sistema de ficheros debe ser un disco flexible, y por otro, el parámetro que indica el tamaño del *ramdisk* debe ser mayor de cero. Estos dos parámetros se pueden establecer de diferentes formas, pero se verán las dos más comunes.

Por una parte, este hecho se puede indicar modificando la propia imagen del núcleo, mediante el comando "rdev":

```
rdev -R minucleo 0
rdev -r minucleo /dev/fd0
```

Con estos dos comandos, se puede "marcar" la imagen de arranque del núcleo, de forma que contendrá esta información sin necesidad de que se la pase ningún cargador.

El segundo método es precisamente el opuesto, puesto que mediante parámetros que se leen durante el arranque, el núcleo puede modificar su comportamiento en la inicialización. Suponiendo a LILO como cargador, las siguientes líneas cumplirían esta función:

```
time = 0
boot = /dev/fd0
ramdisk = 1440  # Tamaño de ramdisk
#
image = minucleo
  root = /dev/fd0 # El floppy es raíz
  label = Linux
```

## ARRANQUE DEL NÚCLEO

Tras modificar los parámetros, o la misma imagen, y rearrancar el PC, el núcleo pasará a ejecutar el código que reconozca estos cambios. Si se modifica la imagen de Linux con '*rdev*', los cambios se graban en las últimas posiciones de la cabecera de 512 bytes del fichero imagen, en la zona formada por *arch/i386/boot/bootsect.S*. Como ya se

comentó hace un par de meses, este código acaba situándose en la posición 90000 (en hexadecimal), y es leído cuando el núcleo pasa a ejecutar en modo protegido, con el código del fichero *arch/i386/kernel/setup.c*. Aquí se inicializan dos variables globales a todo el sistema, *ramdisk_size* y *ROOT_DEV*, respectivamente. Adicionalmente, en este mismo código se recogen los parámetros pasados al núcleo por el cargador, y se devuelven mediante un puntero al *buffer command_line*, que contiene el valor de los mismos.

A partir de este momento, ejecuta el código en *init/main.c*, analizando los valores de los parámetros e inicializando todas las estructuras y *drivers*, según la rutina *start_kernel*. Desde aquí se podrán encontrar en la cadena de parámetros las directivas para modificar aquellos que interesan hoy, a saber:

- *Tamaño del disco* (ramdisk = nnn). Se analiza en la rutina *ramdisk_setup*, y se asigna el valor indicado en el parámetro en la variable *ramdisk_size*.
- *Dispositivo raíz* (root=/dev/xxx). En la rutina *parse_options* se obtiene y convierte el string que se pasa en el parámetro a una pareja de bytes con el código mayor y menor del dispositivo, y este valor se guarda en la variable *ROOT_DEV*.

Como se puede observar, los valores grabados originalmente en el núcleo pueden ser modificados por los parámetros que el cargador pasa, ya que son almacenados en las variables correspondientes en una etapa posterior de la inicialización. Así se confirma lo que ya se indica en la documentación del cargador LILO, es decir, que los parámetros de éste prevalecen sobre los valores pregrabados en el núcleo.

Una vez que, de un modo o de otro, se establecen los valores que indicarán al núcleo que cargue un disco RAM, se seguirá con detalle el proceso en que se desemboca.

**LISTADO 1**

```
long rd_init(long mem_start, int length)
{
        int     i;
        char    *cp;

        if
(register_blkdev(MEM_MAJOR,"rd",&rd_fops)) {
                printk("RAMDISK: Unable to
get major %d.\n", MEM_MAJOR);
                return 0;
        }
        blk_dev[MEM_MAJOR].request_fn =
DEVICE_REQUEST;
        rd_start = (char *) mem_start;
        rd_length = length;
        cp = rd_start;
        for (i=0; i < length; i++)
                *cp++ = '\0';

        for(i=0;i<2;i++) rd_blocksizes[i] = 1024;
        blksize_size[MAJOR_NR] = rd_blocksi-
zes;

        return(length);
}
```

## PREPARACIÓN DEL DISCO RAM

Durante la inicialización de estructuras de Linux en el procedimiento *start_kernel*, se llama a la rutina *blk_dev_init*. Este código se encarga de preparar el entorno del sistema, sus variables y estructuras, para los manejadores de dispositivos de bloques, pero sin activar el hardware asociado. A grandes rasgos, se activan tanto los discos duros IDE, como la unidad de discos flexibles, los CD-ROM no SCSI, y, por último, el disco RAM.

**El disco RAM simula un dispositivo de bloque agrupando la memoria en items de 512 bytes**

En todo este proceso, cada dispositivo puede reservar espacio de memoria que quedará incorporado a los datos del núcleo de forma estática e irreversible. Las variables *memory_start* y *memory_end* son pasadas a cada procedimiento de inicialización de dispositivos, de forma que éste puede modificarlas. En el caso del disco de memoria, se invoca a *rd_init*, cuyo código se muestra en el listado 1. Básicamente se trata de reservar un espacio en memoria, del tamaño en Kbytes indicado por la variable *ramdisk_size*.

Una vez completada la inicialización de todos los dispositivos, del sistema de ficheros y del gestor de memoria, Linux está listo para comenzar a funcionar. Como ya se ha comentado, la llamada *fork* duplica la imágen original del núcleo, que mantendrá la funcionalidad monolítica del mismo en memoria, mientras que la nueva formará el primer proceso dinámico del sistema.

Con estas dos entidades, se pueden introducir dos conceptos que serán útiles siempre que se estudie un sistema operativo con posibilidad de ejecutar en varios niveles de privilegio (protección). Se denomina "lado claro" (*user space*) de una llamada al sistema al código de librería que se enlaza al programa y que ejecuta sin privilegios, preparando parámetros justo antes de invocar la función del núcleo (de realizar la conmutación hardware que el procesador debe hacer para cambiar el nivel de ejecución). El "lado oscuro" (*kernel space*) es el código que el núcleo sigue como consecuencia de una llamada al sistema, con nivel total de privilegio, desde que se entra en una "trap" hasta que se retorna de nuevo al programa de usuario. Obsérvese la figura 1.

La escisión que se realiza tras el *fork* inicial de Linux hace que el monolito quede como "lado oscuro", mientras que este primer proceso, la copia, sea ejecutado desde el "lado claro". Toda esta graciosa terminología de "guerra de las galaxias" pretende distinguir que a partir de ahora, en el núcleo, las rutinas del tipo *sys_** serán los puntos de entrada que el núcleo ofrece desde el lado oscuro, y que serán los que desencadenen la ejecución del mismo, como trozos de código "privilegiado" que corren como extensiones del programa del lado claro. Por ejemplo, existe un punto de entrada denominado *sys_mount*, que satisface las peticiones de la llamada al sistema *mount*, realizada por los procesos. El acceso a las unidades de almacenamiento se producirá por la ejecución normal de los programas de usuario, a traves de las funciones *sys_**.

La primera llamada al sistema, que se realiza una sola vez durante toda la vida del mismo, se denomina *sys_setup*, y permite completar la ini-

cialización del sistema de ficheros desde el lado claro, montando la unidad de almacenamiento que será la raíz de éste. Esta unidad no se podrá ya desmontar mientras haya algún proceso Unix ejecutando. Esta primera llamada tiene el cuerpo de su código en el fichero *fs/filesystems.c*, y se encarga de llamar a *device_setup* y a *mount_root*.

Aquí se produce un "despertar" en los dispositivos de bloque: Se reconocen las particiones, y se testea si el medio de almacenamiento es legible. A diferencia de la inicialización (rutinas **_init*), aquí se accede ya físicamente al hardware para comenzar el trabajo. Entre otras cosas, en el caso de la inicialización del disco RAM se cede control a *rd_load*, que copia los bloques del disco a memoria, en la zona que *rd_init* reservó, y trata de averiguar el formato del sistema de ficheros contenido en ellas, mediante la lectura del primer bloque.

Como se observa en el listado 2, en el núcleo 1.2.13 existe la posibilidad de acceder a una unidad con formato *minix* o *ext2*. Si este reconocimiento tiene éxito, la variable *ROOT_DEV* se modifica, para indicar que la unidad de discos flexibles, que era la raíz inicial, ahora tiene su contenido copiado en RAM, que será el dispositivo raíz, y que el modo de acceso a la misma es de lectura y escritura, independientemente de lo que indicase el usuario en origen.

Cuando *sys_setup* sigue su ejecución, se invoca a *mount_root*. Esta función, situada en *fs/super.c*, se encarga de asociar el sistema de bloques de la unidad *ROOT_DEV* al sistema de i-nodos del sistema de ficheros. Los *i-nodos* son descriptores lógicos comunes a todos los sistemas de ficheros que Linux soporta (de hecho es una denominación común de los recursos de disco en cualquier sistema Unix), mediante el metasistema de ficheros VFS. La figura 2 muestra un esquema del proceso visto aquí.

Al montar una unidad, se lee el primer bloque (un i-nodo que se denomina superbloque, y que usualmente contiene una información descriptiva de la unidad, como su formato, capacidad, localización de recursos, etcétera). Cuando se abre un fichero, por ejemplo, se deja marcado en memoria el i-nodo que contiene la entrada de directorio que contiene información sobre dicho fichero, y así. El i-nodo del superbloque siempre estará marcado como ocupado mientras la unidad asociada esté montada.

**LISTADO 2**

```
static void do_load(void)
{
        struct buffer_head *bh;
        struct super_block {
          union
          {
            char minix [sizeof (struct
minix_super_block)];
            char ext2 [sizeof (struct
ext2_super_block)];
          } record;
        } sb;
        struct minix_super_block *minixsb =
                (struct minix_super_block
*)&sb;
        struct ext2_super_block *ext2sb =
                (struct ext2_super_block
*)&sb;
        int                     block, tries;
        int                     i = 1;
        int                     nblocks;
        char                    *cp;

        /*
         * Check for a super block on the dis-
kette.
         * The old-style boot/root diskettes had
their RAM image
         * starting at block 512 of the boot dis-
kette.  LINUX/Pro
         * uses the entire diskette as a file sys-
tem, so in that
         * case, we have to look at block 0.  Be
intelligent about
         * this, and check both... - FvK
         */
        for (tries = 0; tries < 1000; tries += 512)
{
                block = tries;
                bh =
breada(ROOT_DEV,block+1,BLOCK_SIZE, 0,
PAGE_SIZE);
                if (!bh) {
                        printk("RAM-
DISK: I/O error while looking for super block!\n");
                        return;
                }

                /* This is silly- why do we
require it to be a MINIX FS? */
                *((struct super_block *) &sb)
=
                        *((struct
super_block *) bh->b_data);
                brelse(bh);


                /* Try Minix */
                nblocks = -1;
                if (minixsb->s_magic ==
MINIX_SUPER_MAGIC ||
                    minixsb->s_magic ==
MINIX_SUPER_MAGIC2) {
                        printk("RAM-
DISK: Minix filesystem found at block %d\n",

block);
                        nblocks =
minixsb->s_nzones << minixsb->s_log_zone_size;
                }

                /* Try ext2 */
                if (nblocks == -1 && (ext2sb-
>s_magic ==

EXT2_PRE_02B_MAGIC ||
                        ext2sb->s_magic
== EXT2_SUPER_MAGIC))
                {
```

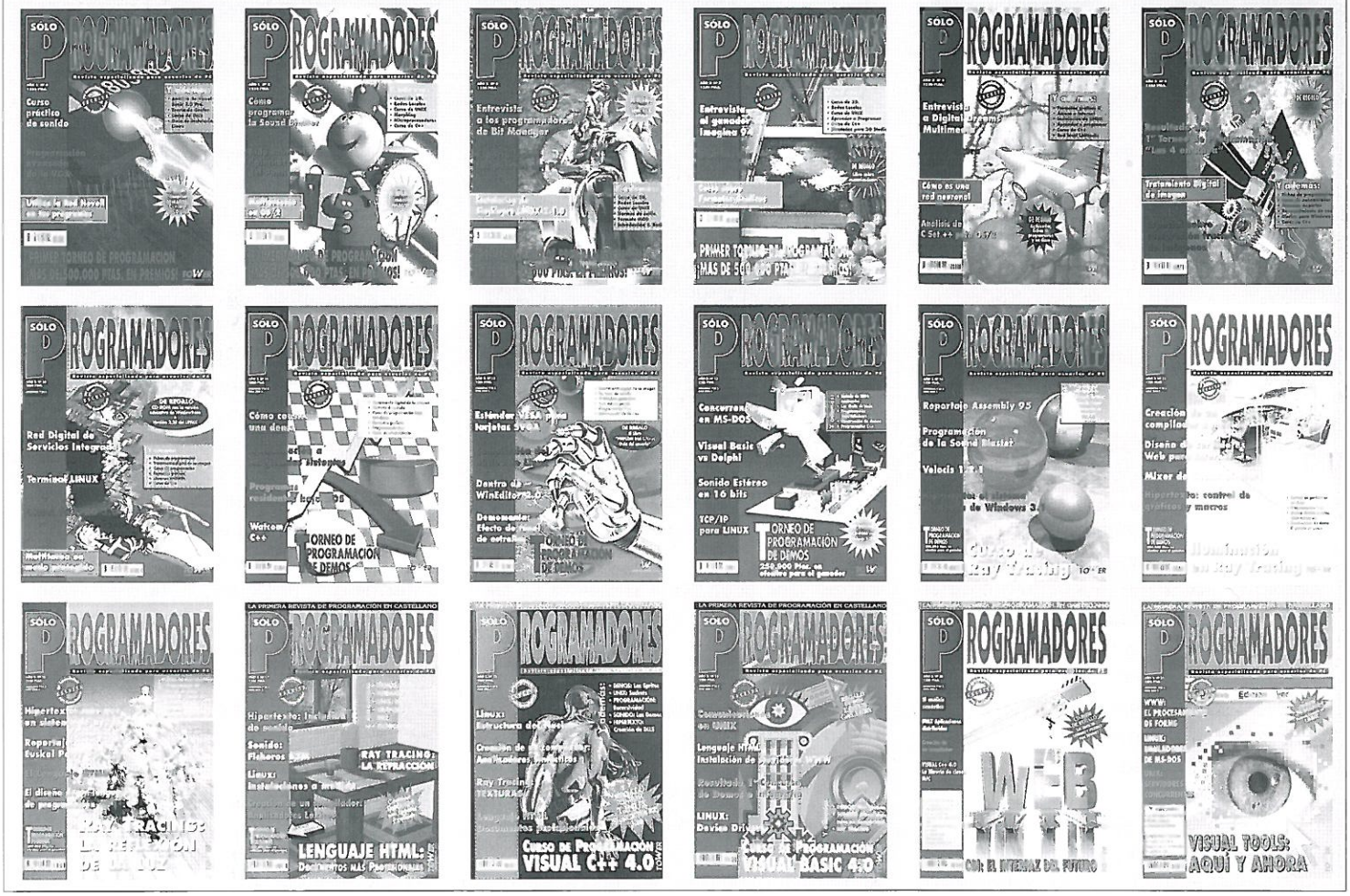
```
                        printk("RAM-
DISK: Ext2 filesystem found at block %d\n",

block);
                        nblocks =
ext2sb->s_blocks_count;
                }

                if (nblocks == -1)
                {
                        printk("RAM-
DISK: trying old-style RAM image.\n");
                        continue;
                }

                if (nblocks > (rd_length >>
BLOCK_SIZE_BITS)) {
                        printk("RAM-
DISK: image too big! (%d/%d blocks)\n",

nblocks, rd_length >> BLOCK_SIZE_BITS);
                        return;
                }
                printk("RAMDISK: Loading
%d blocks into RAM disk", nblocks);

                /* We found an image file
system.  Load it into core! */
                cp = rd_start;
                while (nblocks) {
                        if (nblocks > 2)
                                bh = brea-
da(ROOT_DEV, block, BLOCK_SIZE, 0,
PAGE_SIZE);
                        else
                                bh =
bread(ROOT_DEV, block, BLOCK_SIZE);
                        if (!bh) {

printk("RAMDISK: I/O error on block %d, abor-
ting!\n",

block);

return;
                        }
                        (void)
memcpy(cp, bh->b_data, BLOCK_SIZE);
                        brelse(bh);
                        if (!(nblocks— &
15)) printk(".");

                        cp +=
BLOCK_SIZE;
                        block++;
                        i++;
                }
                printk("\ndone\n");

                /* We loaded the file system
image.  Prepare for mounting it. */
                ROOT_DEV =
((MEM_MAJOR << 8) | RAMDISK_MINOR);
                return;
        }
}
```

listado puede ir en tabla de 2 columnas

# CÓMO SUSCRIBIRSE A

SÓLO PROGRAMADORES

Revista práctica para usuarios de PC



**Suscríbase enviando este cupón por correo o fax (91) 661.43.86, o llamando al teléfono (91) 661.42.11 Horario 9 a 14 y 15:00 a 18:00 h.**

**Deseo suscribirme a la revista SÓLO PROGRAMADORES acogiéndome a la siguiente modalidad:**

☐ Suscripción: 1 año (12 números) **por sólo** 11.950 ptas. (ahorro 20%). ☐ Estudiantes carreras técnicas: 8.950 ptas. (ahorro 40%)

*** ESTA OFERTA ANULA LAS ANTERIORES, DESCUENTOS NO ACUMULABLES.**

Nombre y apellidos.....Domicilio.....

Población.....C.P .....Provincia.....Telf.....Profesión.....

**FORMA DE PAGO:**

☐ Con cargo a mi tarjeta VISA nº ____ ____ ____ ____

Fecha de caducidad de la tarjeta.....Nombre del titular, si es distinto.....

☐ Domiciliación bancaria.

Señor Director del banco .....

Población .....

Ruego a vd. que se sirva cargar en mi ☐ cuenta corriente ☐ libreta

e ahorro número.....

recibo que le será presentado por TOWER COMMUNICATIONS, S.R.L.

omo pago de mi suscripción a la revista SÓLO PROGRAMADORES.

CODIGO CUENTA CLIENTE

| ENTIDAD | OFICINA | DC | Nº CUENTA |
|---|---|---|---|
| | | | |

**Firma:**

☐ Contra-reembolso del importe más gastos de envío.

☐ Cheque a nombre de TOWER COMMUNICATIONS S.R.L., que adjunto.

☐ Giro Postal (adjunto fotocopia del resguardo).

Rellena este cupón y envíalo a:
TOWER COMMUNICATIONS SRL
C/ Aragoneses, 7
28100 Pol. Ind. ALCOBENDAS (Madrid)

Dado que cualquier proceso que esté ejecutando está asociado a un fichero abierto en disco, y que por tanto está ocupando i-nodos en memoria, se llega a una curiosa paradoja, siempre refiriéndonos al núcleo 1.2.13. Mientras un sólo proceso ejecute, hay i-nodos ocupados de una unidad, y por tanto no se puede desmontar. Pero, para desmontar dicha unidad, se debe invocar con la primitiva *umount*, desde el lado claro, por un proceso en ejecución.

Es decir, que la unidad raíz no se puede desmontar, si no hay una función especial en el lado oscuro que tras ser invocada por un proceso, sea capaz de:

- cerrar los descriptores de ficheros del sistema original (liberar i-nodos)
- desmontarlo
- montar el nuevo dispositivo
- reabrir los descriptores en este último

A partir de entonces se podría devolver control al proceso que esté ejecutando, y todo seguiría su curso normal. Esta capacidad está disponible a partir de la versión 2.0 de Linux, de inminente aparición. Sin embargo, si se monta un disco RAM con el núcleo actual, ya no se podrá desmontar (y liberar la memoria que ocupa) mientras Linux ejecute.

En las versiones 1.3.xx (el desarrollo intermedio entre 1.2.13 y 2.0) aparece una nueva idea muy útil que motivó este cambio de comportamiento. Si se pudiese montar el disco RAM sólo durante la inicialización, se podrían realizar operaciones de inicio del sistema, cargar algunas partes del núcleo (los drivers del hardware que se haya detectado en arranque) y desechar el resto, ahorrando espacio final. Una vez cargados los fragmentos necesarios, el disco RAM se destruye, reusando una unidad de disco duro. Dicho de otra forma, se pueden descargar partes del monolito Linux, que ocupa memoria RAM y no se puede paginar, y reutilizarlas como drivers cargables desde el lado claro. Esto resulta en un desmenuzado del núcleo, aproximando la idea a la de "núcleo por capas" que acompaña al diseño de sistemas según las tendencias más modernas.

**LISTADO 3**

```
static void do_rd_request(void)
{
        int     len;
        char    *addr;

repeat:
        INIT_REQUEST;
        addr = rd_start + (CURRENT->sector
<< 9);
        len = CURRENT->current_nr_sectors <<
9;

        if ((MINOR(CURRENT->dev) != RAM-
DISK_MINOR) ||
            (addr+len > rd_start+rd_length)) {
                end_request(0);
                goto repeat;
        }
        if (CURRENT-> cmd == WRITE) {
                (void ) memcpy(addr,
                                CURRENT-
>buffer,
                                len);
        } else if (CURRENT->cmd == READ) {
                (void) memcpy(CURRENT-
>buffer,
                                addr,
                                len);
        } else
                panic("RAMDISK: unknown
RAM disk command !\n");
        end_request(1);
        goto repeat;
}
```

**La función de atención de dispositivo.**

Visto de otra forma, una máquina sin disco duro (*diskless workstation*) podría llegar a ser cargada completamente por red ethernet (o por línea serie, al menos en teoría), transfiriendo el núcleo y un disco RAM inicial, que sería sustituido por un disco de red. Otra aplicación podría ser tener Linux en EPROM, y construir la imagen en memoria a partir de éste para empezar a trabajar.

Resumiendo, el disco RAM en las nuevas versiones de Linux se usa como una lanzadera que almacena varios drivers desechables mientras el sistema arranca de forma normal. Por contra, el disco de la versión 1.2.13, si se usa, se monta de forma irreversible en el sistema, y está pensado para su uso con discos de emergencia o instalación de Linux.

## LA RUTINA DE ESTRATEGIA

Una vez que el disco está inicializado y relleno de los datos adecuados, el sistema accede al mismo para leer y escribir en un dispositivo de bloques simulado. Para ello, esta categoría de periféricos utilizan una función especial, invocada desde el sistema de ficheros, que se encarga de la transferencia física de los datos. El dispositivo que nos ocupa hoy es trivial, puesto que el acceso al mismo se basa en copias de memoria a memoria (entre la que se ha reservado para el disco y la que corresponde al buffer del usuario que solicita la lectura o la escritura).

Se ha de tener en cuenta que esta rutina es crítica en el tiempo (se la invoca con las interrupciones inhibidas) y suele ser la principal responsable del rendimiento del manejador de cada dispositivo concreto. Según se observa en el listado 3, la rutina *rd_request* marca la disponibilidad para atender eventos (con la macro *INIT_REQUEST*, llamada por todos los dispositivos de bloque), y queda esperando un evento. Linux abstrae al manejador de algunos detalles feos, y le gestiona una cola donde *CURRENT* indica la operación actual. Se calcula el tamaño y posición de la operación (*sector* indica el posicionamiento, y *current_nr_sectors* la longitud de información a mover) en *addr* y *len*. Si se pregunta por qué aparecen operaciones de rotado con nueve bits, la respuesta es que el tamaño de bloque de este dispositivo es de 512 bytes (2 elevado a nueve).

Tras una comprobación de que la petición se hace sobre el disco adecuado, y que no se sobrepasa su tamaño físico (*rd_length*), se efectúa la misma. Se admiten en este manejador operaciones de lectura (*cmd == READ*) o escritura (*cmd == WRITE*), y se copia la información a donde indique la petición (*buffer*). Finalmente, se indica al núcleo que la operación se ha completado (*end_request*) y se vuelve a empezar. Si todo esto le ha parecido excesivamente complicado, consuélese pensando que es una de las partes más oscuras y complicadas de todo el núcleo, y que más cuestan manejar. Al fin y al cabo, no hay tantos dispositivos de bloque en Linux, y son pocos los "gurús" en este campo.

# HERRAMIENTAS DE DEPURACIÓN

*Jorge R. Regidor*

**En esta entrega del curso de programación para Windows se va a tratar un tema de gran interés práctico, como son las herramientas de depuración de errores de aplicaciones que ofrece Visual C++ 4.0, para ya en próximos números centrarnos en el tema de programación de controles de windows 95.**

Después de seis entregas del curso de programación para Windows con Visual C++ 4.0 y su librería de clases MFC, llega el momento de dedicar un artículo a tratar de conocer una de las herramientas básicas de todo entorno de desarrollo, como es el depurador de código.

Cuando comenzaba a dar mis primeros pasos en la programación, leí sin darle mucha importancia, que aproximadamente el ochenta por ciento del tiempo de desarrollo de una aplicación se invertía en la depuración de la misma. A día de hoy y después de haber introducido bastantes líneas de código, me doy cuenta de la veracidad de aquella afirmación, y hasta incluso que en algunos tipos de aplicaciones nos acercamos claramente al noventa y tantos por ciento. Por eso, cualquier mejora en las herramientas de depuración afecta de modo notable al rendimiento en la elaboración de aplicaciones.

**Aproximadamente el ochenta por ciento del tiempo de desarrollo de una aplicación se inverte en depurarla**

## ¿CUÁNDO HAY QUE UTILIZAR EL DEPURADOR Y CUÁNDO NO?

Parece que las herramientas de depuración debieran ser utilizadas en todo momento y a pleno rendimiento, pero existen casos en los que su utilización debe limitarse o bien sencillamente no puede usarse.

Para ilustrar esto se propone un ejemplo práctico. Supongamos un sistema de control a tiempo real, donde se debe enviar una señal de control en función de una señal leída continuamente. Si en este escenario se activa la depuración paso a paso para poder evaluar los valores de las variables, la pila o el estado de los registros, el resultado es que al parar la ejecución se abre el lazo de control y el sistema evoluciona de modo catastrófico. En este caso se deberá utilizar otro tipo de depuración basado en trazas en el código fuente, que sean mostradas según se pasa por ellas y que muestren los datos necesarios para conocer el estado de la aplicación a su paso por ese punto.

Por norma, el uso del depurador mediante la ejecución paso a paso puede utilizarse cuando las paradas no influyen sobre la correcta ejecución de la aplicación. Por otra parte, y en caso de que se cumpla la condición anterior (en la gran mayoría de las veces), la utilización de las herramientas de depuración deben ser norma de habitual cumplimiento. Seguramente con estas herramientas de depuración se podrán ver errores que, a priori, y sobre el código fuente, no parecían existir.

## HERRAMIENTAS DE DEPURACIÓN EN VISUAL C++

La propia metodología de programación en C++ para Windows con la librería MFC, y las herramientas como *ClassWizard* o *ClassBar*, forman la primera línea de prevención de errores, ya

que sriven de guía al crear el esqueleto o la base sobre la cual se debe añadir el código. Esta base está formada por una cantidad de objetos ya debidamente depurados por el propio equipo de *Microsoft*, lo que liberará al desarrolador de esa labor. Además, y gracias a la por norma buenísima documentación que Visual C++ 4.0 ofrece sobre el método de utilización de las clases y sus métodos, la probabilidad de introducir errores de codificación se reduce notablemente.

Visual C++ 4.0 incluye varias herramientas para facilitar la tediosa búsqueda de errores de ejecución en las aplicaciones, entre las que se van a tratar *Debug* y MFC *Tracer*.

## DEBUGGER

El primer paso que hay que hacer para utilizar el depurador en la aplicación es activar el modo *Debug* al compilar el programa. Para ello, hay que activar la opción *Win Debug* de la lista en la barra de controles del proyecto. Después de esto, se puede cambiar la configuración del *debugger*. Para realizar esta operación hay que abrir la opción *Settings...* del menú *Build* o bien pulsar las teclas <Alt+F7>. Una vez que se muestra el diálogo de configuración del proyecto, se debe seleccionar la configuración *Win32 Debug* de la *TreeList* de la izquierda (Ver Figura A). Ahora, se selecciona el *tab* llamado *Debug*, donde pueden verse todas las opciones de depuración.

Dentro de este *tab* se pueden diferenciar dos categorías de información elegidas desde la lista de la parte superior. En la categoría general se pueden seleccionar nombres y directorios, y en la categoría *Additional DLLs* se deben incluir todas las DLLs necesarias para que la aplicación pueda ejecutarse.

**Al compilar el programa se debe activar el modo debug antes de utilizar el depurador**

## CATEGORÍA GENERAL

En esta categoría se observan varios campos. En el primero, *Executable For Debug Session*, se debe indicar el nombre del ejecutable a ejecutar por el debugger. En el segundo campo, *Working Directory*, se debe especificar el directorio donde se va a efectuar la depuración. Si no se indica nada, se realizará en el directorio donde se encuentre el ejecutable. Normalmente no hace falta cambiar estos campos, ya que al crearse el proyecto se almacenan los valores adecuados. El siguiente campo, *Program Arguments*, permite introducir los argumentos que se le pasarán al programa al empezar su ejecución desde el *debugger*. Por último, en el campo *Remote Executable Path and File Name*, se puede indicar el nombre y la ruta del ejecutable para la depuración remota.

## CATEGORÍA ADDITIONAL DLLs

En la categoría *Additional DLLs* se pueden incluir todas las librerías de enlace dinámico necesarias para la ejecución de la aplicación. Para ello sólo hay que pulsar el botón para añadir nuevas DLLs e indicarle la ruta donde se encuentra.

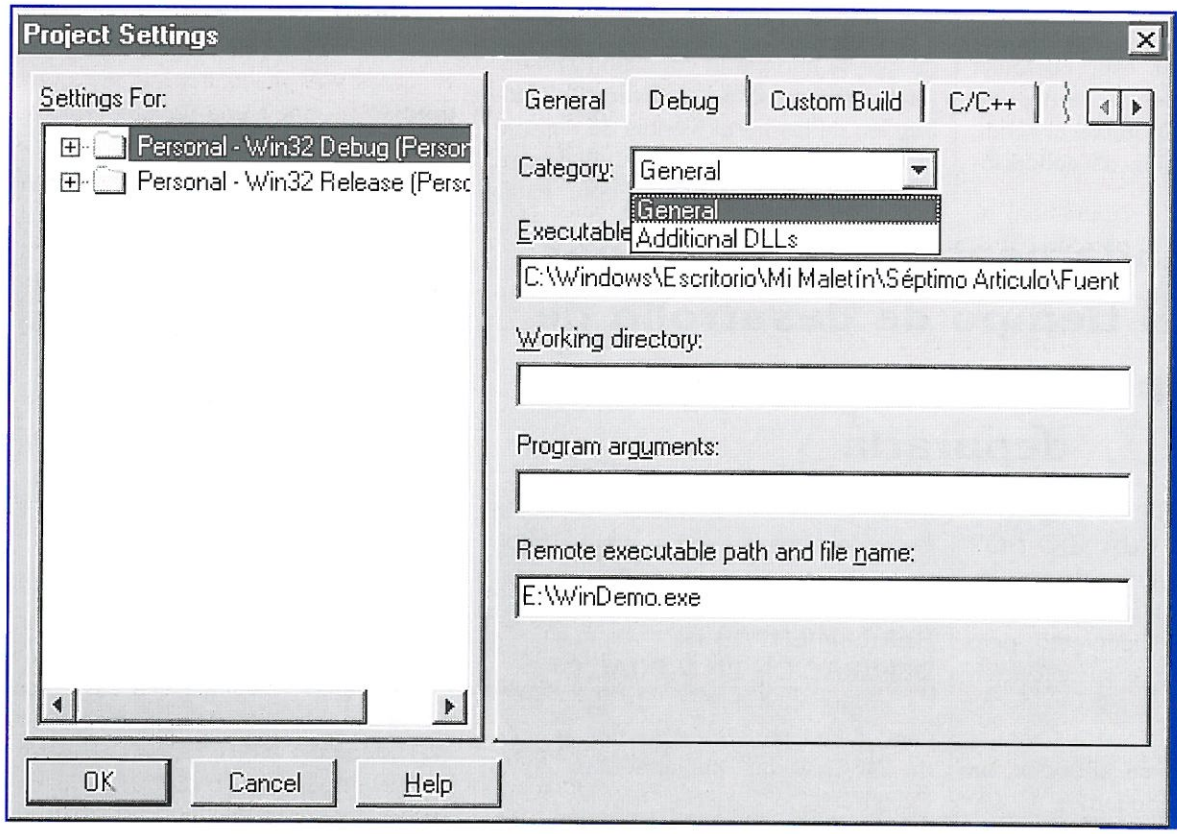



**Figura A: Configuración del depurador de código.**

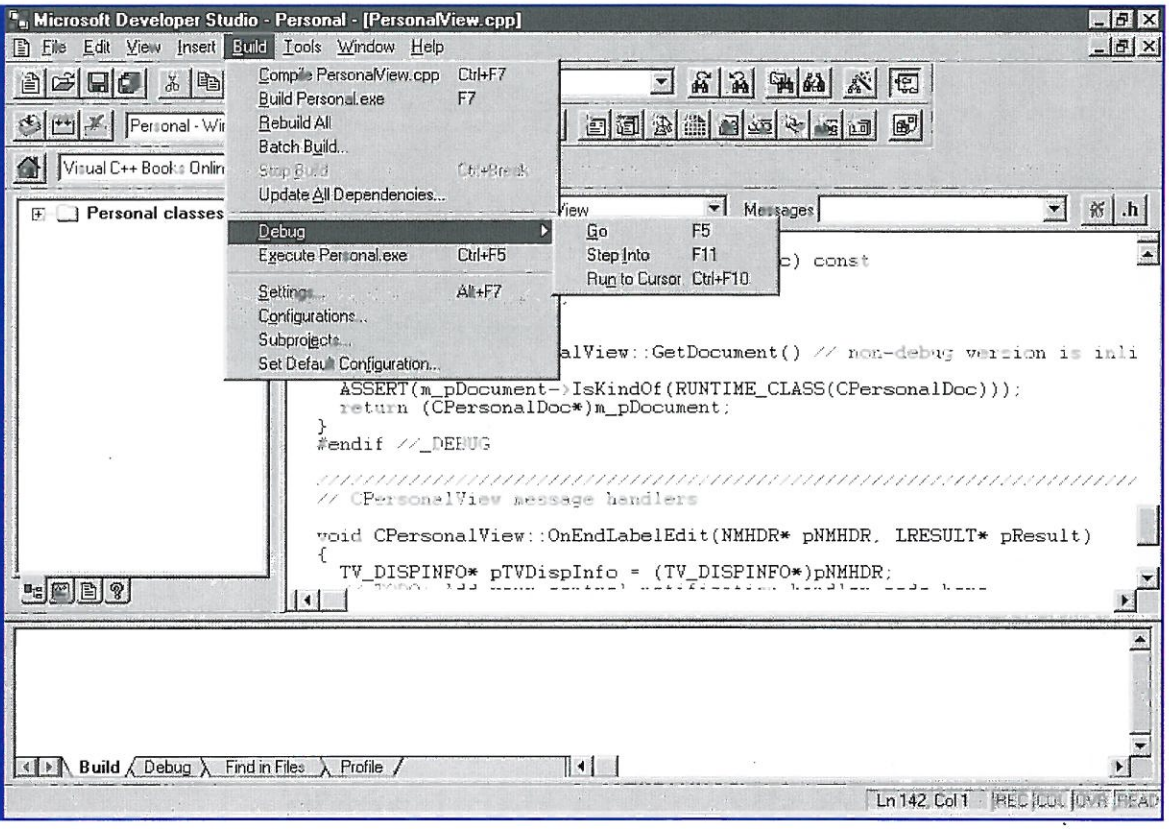



**Figura B: Menú de inicio de ejecución con el depurador.**

## INICIAR LA DEPURACIÓN

Para iniciar la depuración se debe seleccionar la opción Debug del menú *Build*. De aquí se ven tres opciones, *Go* (F5) que permite ejecutar el programa sin pausas, *Step Into* (F11) que inicia la ejecución paso a paso, y *Run to Cursor* (CTRL+F10), que ejecuta el programa hasta que el flujo de ejcución llega hasta la posición actual del cursor en el código fuente (Ver Figura B).

Una vez en modo de depuración, el aspecto del entorno de desarrollo cambia de aspecto, de tal modo que la ventana de visualización del proyecto, *Proyect Workspace*, desaparece, y aparecen dos ventanas en la parte inferior que permiten saber el estado de las variables de la aplicación (Ver Figura C), así como ver las expresiones que se deseen. Además, aparece una barra de herramientas con todos los botones para utilizar el *debugger*. Dentro de esta barra se pueden activar todas las ventanas que ofrecen un visión diferente del estado del programa. En las siguiente líneas se van a tratar cada una de ellas.

**Para moverse paso a paso por el ejecutable existen cuatro botones en la barra de herramientas del debugger**

## VARIABLES WINDOW

Esta ventana visualiza de forma automática el estado de las variables de la aplicación. Para ello tiene tres secciones que muestran las variables en uso actualmente, las variables locales a la función donde nos encontramos y el contenido del puntero *this*.

Cuando la variable que se muestra es una estructura, se muestra el contenido de ésta mediante un árbol, que permite expandir sus campos y visulizarlos detalladamente.

En la parte superior de la ventana aparece una *listbox* donde se encuentran todas las funciones activas en cada momento, de tal forma que pueden seleccionarse y así observar sus propias variables locales.

## WATCH WINDOW

Esta ventana contiene cuatro *tabs* donde se pueden visualizar las distintas variables y expresiones que se deseen trazar. Para añdir una variable en esta ventana hay que seleccionar el *tab* que se quiera, y en una fila libre introducir su nombre. Después de esta operación, se puede observar el valor de la variable o expresión introducida.

Existe un segundo método para incluir variables y expresiones. Para ello habrá que ir al código fuente y seleccionar la expresión, después de esto se pulsa el botón izquierdo del ratón y, manteniéndolo pulsado, se mueve hasta la ventana. En definitiva, una operación de *drag and drop*.

### *QUICK WATCH*

Esta utilidad permite visualizar el contenido de una variable o una expresión de forma más rápida. Para ello basta con situar el cursor sobre la expresión y pulsar <SHIFT+F9> o pulsar el botón correspondiente de la barra de herramientas de depuración. Se mostrará

una ventana donde se indicará el valor de la expresión y un botón por si se desea añadirla a *Watch Window*.

Además, existe otro método aún más rápido de ver el valor de una expresión. Para ello basta con mover el ratón sobre la expresión y pararlo. En ese momento aparecerá un pequeño mensaje al lado del cursor que visualizará el valor de la expresión. Esta utilidad sólo funciona cuando está la ejecución en pausa.

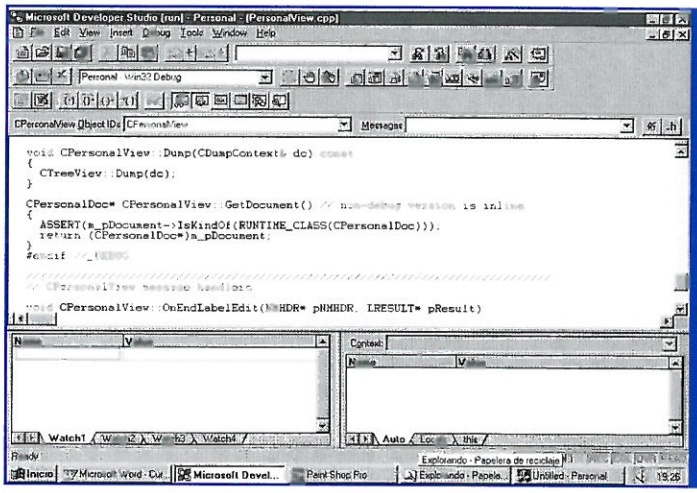


Figura D: Aspecto del entorno de desarrollo en depuración

## REGISTER WINDOW

Esta ventana visualiza el estado de los registros del microprocesador y del coprocesador matemático. Se pueden ver el contenido de EAX, EBX, ECX, etc; así como los registros de segmento, las banderas, puntero de instrucciones (IP). Existe una opción para habilitar o no los registros del coprocesador matemático. Para ello se pulsa el botón derecho sobre la ventana y aparece un menú con las opciones de esta ventana.

## MEMORY WINDOW

En esta ventana se puede visualizar e incluso editar el contenido de un área de memoria. Para ello sólo se debe indicar la dirección de 32bits donde se quiere empezar a observar. Existen tres modos de visualización, una orientada al byte (8 bits), donde se muestra el contenido byte a byte y sus ASCIIs correspondientes. La segunda está orientada al *short* (16 bits) y la tercera orientada al *long* (32 bits). Si se observa en el contenido de memoria y se visualiza en sus tres formatos, se verá algo parecido a lo siguiente:

*Dirección: 0x00400000*
*Contenido (Modo Byte): 4D 5B 90 00*
*Contenido (Modo Short): 5B4D 0090*
*Contenido (Modo Long): 00905B4D*

**Es posible realizar una depuración remota, bien por un puerto serie o por red con TCP/IP**

Esta variación en el orden entre los diferentes modos es debido a la forma que tienen los microprocesadores de Intel de almacenar los datos en memoria, los bytes de menor peso en direcciones más bajas.

## CALLSTACK WINDOW

Esta ventana permite visualizar todas la lista de funciones que llevan hasta la función actual en curso, la cual aparece en la parte superior de la ventana (Ver Figura D). Cada función aparece por defecto con la lista de sus parámetros y el tipo de los mismos, pero de igual modo pueden suprimirse abriendo el menú flotante con el botón derecho del ratón. Si se situa el cursor sobre una de las funciones y se pulsa un doble click, se abre la ventana donde se muestra el código fuente de esa función.

## DISASEMBLY WINDOW

En esta ventana se muestra el ejecutable desensamblado con instrucciones directas al microprocesador. Además, se muestra entrelazado a éste el código fuente en C++ que lo compone. También se puede activar que se muestre el código máquina de cada instrucción en ensamblador.

## MOVIÉNDONOS POR EL CÓDIGO FUENTE

Para moverse paso a paso por el ejecutable, existen cuatro botones en la barra de harramientas del *debugger* (Ver Figura E) que permiten realizar

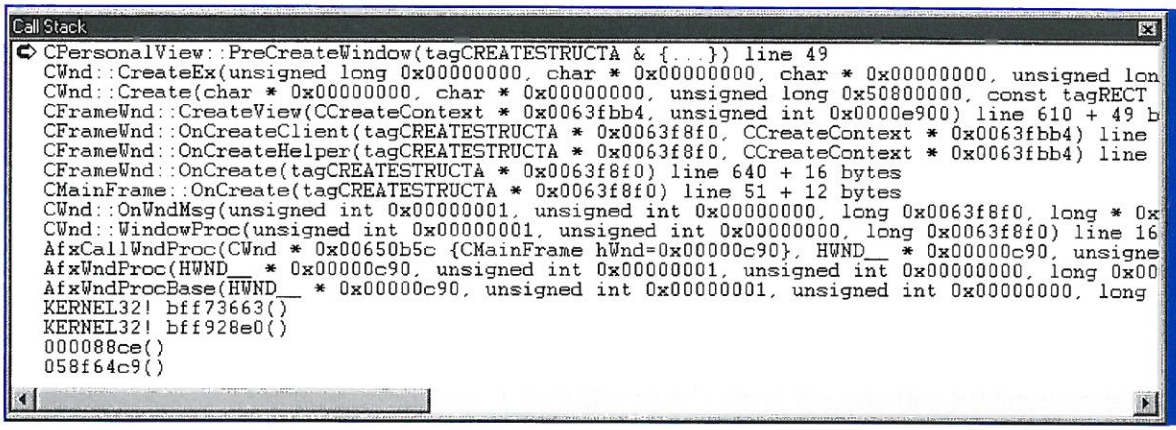



Figura D: Ventana Callstack.

**Figura E: Barra de herramientas de depuración.**

diversos saltos en función de cada situación. Estos saltos son *Step+Into*, que permite entrar dentro de una función y ejecutar paso a paso en ella; *Step+Over*, que al llegar a una función la ejecuta de un paso; *Step+Out*, que permite ejecutar de modo continuo hasta salir de la función actual y *Run to Cursor*, que ejecuta el programa hasta la posición actual del cursor.

Además, se pueden poner puntos de parada o *BreakPoint* en cualquier parte del código fuente. De tal forma que al ejecutar el programa sólo se parará si pasa por dicho punto de parada. Para activar y desactivar un punto de parada hay que situarse sobre la línea donde se quiere situar y pulsar el botón de *breakpoint* o bien pulsar <F9>. Para desactivarlo se hace la misma operación. Además existe otro botón que permite eliminar todos los puntos de parada dentro del ejecutable.

## REMOTE DEBUGGER

Esta utilidad permite ejecutar el programa en un ordenador y la depuración en otro. La comunicación entre ellos se puede realizar por un puerto serie o por red con el protocolo TCP/IP.

Para activar esta funcionalidad hay que ejecutar en el equipo remoto la utilidad *MSVCMON.EXE* contenida en el directorio msdev\bin. Además hay que tener una copia del programa a depurar. Al ejecutar esta utilidad, se muestra una ventana donde se puede elegir entre LAN o puerto serie. También hay un botón (*Settings*) que permite configurar la comunicación. En el caso de un puerto serie, se permite seleccionar la velocidad de transmisión y el protocolo de control de flujo entre *XON/XOFF* y *Hardware*. Al terminar la configuración se pulsa *connect* y la utilidad se queda esperando órdenes de la máquina remota con el *debugger* de Visual C++.

La configuración en la máquina donde se va a realizar la depuración es la siguiente. Hay que abrir la opción *Remote Connection* del menú *Tools* y aparecerá una ventana que permite elegir entre depuración en local (la propia máquina), puerto serie o red con el protocolo TCP/IP. Además, también existe un botón de configuración de la comunicación. Para el caso de un puerto serie se debe elegir la misma velocidad y protocolo que en la máquina remota.

Para iniciar la depuración se hace igual que en local, sólo que el programa se ejecuta en la máquina remota, y toda la información de variables, expresiones, memoria, registros, etc..., se visualizarán en la máquina con Visual C++.

**Los mensajes producidos por las trazas se pueden visualizar en el tab DEBUG de la ventana de salida**

## MFC TRACER

Al principio del artículo se decía que hay situaciones donde no se puede parar la ejecución del programa, por lo que la depuración paso a paso era imposible realizarla. En estos casos, y en general en todos se puede realizar una depuración basada en trazas.

La idea es la siguiente: donde se quiera ver el valor de una variable, o que se ha pasado por ahí, se pone una traza y ésta envía un mensaje a *MFC Tracer* que lo muestra y de esa forma informa del estado y evolución del programa.

Para poner una traza basta con llamar a la función *TRACE* que admite parámetros como si de *printf* se tratase. Además, esta función sólo funciona en modo *DEBUG*, por lo que al realizar la versión definitiva (*RELEASE*) no hace falta eliminarlas. El siguiente código ilustra su utilización:

```
hIcon = ::LoadIcon
(hInstance, ICONO_LIBRO);
if(hIcon == NULL)
{
 TRACE("Atención, no se cargó el icono
LIBRO");
}
```

Además de *TRACE*, existe otra función que permite parar la ejecución del programa en el caso de que no se cumpla cierta condición. Supongamos que el cargar el icono LIBRO es fundamental para la aplicación, y por tanto si no se carga correctamente hay que parar la ejecución. En este caso se utilizará la función *ASSERT*, la cual para la ejecución si su parámetro es 0 o NULL. Un ejemplo de esto último sería:

```
hIcon = ::LoadIcon(hInstance,
ICONO_LIBRO);
ASSERT(hIcon);
```

**Figura F: Ventana de salida de mensajes**

Todos los mensajes producidos por las trazas se pueden visualizar en el *tab DEBUG* de la ventana de salida (*Output Window*) (Ver Figura F), que se puede ver al terminar la ejecución del programa, o bien activando la ventana en el menú *View*.

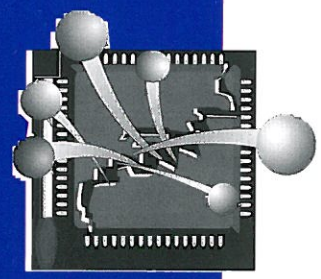
## CONCLUSIÓN

En este artículo se ha pretendido mostrar las herramientas de depuración que Visual C++ 4.0 ofrece. Además se ha querido hacer hincapié en su utilización, ya que cuando existe algún tipo de problema en el programa permiten buscarlo con mayor eficacia y rapidez.

En la próxima entrega se continuará con la introducción de nuevos elementos a las aplicaciones y se tratará el tema de programación de controles de Windows 95, y en concreto del control *Tree View*.

# LOS CONTROLES BÁSICOS (II)

*Juan Manuel y Luis Martín*



**En el artículo del mes pasado se analizaron algunos de los controles básicos disponibles en Visual Basic 4.0. En esta nueva entrega se estudiarán el resto de los controles estándar considerados básicos para la creación de aplicaciones. Una vez que el lector domine su manejo, estará en condiciones de crear sus primeras aplicaciones con Visual Basic**

Como ya se mencionó en entregas anteriores de esta serie, las aplicaciones Windows creadas con Visual Basic se basan en la adición de controles a los formularios, la especificación de sus propiedades y la escritura del código que se ejecutará como respuesta a los distintos eventos que se produzcan. Por tanto, gran parte del dominio de Visual Basic radica en el conocimiento de los controles disponibles y de sus características (propiedades, métodos y eventos).

En el artículo anterior se analizó el funcionamiento de los botones de comando, las etiquetas, los cuadros de texto, las casillas de verificación, los marcos y los botones de opción. En este artículo se completará la descripción de los controles básicos con otros, como las barras de desplazamiento, los temporizadores, los cuadros de lista y los cuadros combinados.

Todos estos controles se encuentran siempre disponibles en la Caja de herramientas de Visual Basic 4.0, independientemente de la edición de éste, ya que se trata de controles estándar. Las clases a partir de las que se crean estos controles se encuentran implementadas en el propio archivo .EXE de Visual Basic. El resto de los controles estándar están relacionados con temas muy específicos (gráficos, Automatización OLE, bases de datos, etc) y serán analizados en posteriores artículos.

## BARRAS DE DESPLAZAMIENTO

Como los usuarios de Windows ya saben, cuando en una aplicación se desea desplazar la información visualizada en un documento o una superficie, se utilizan unos controles denominados Barras de desplazamiento (*Scroll bars*). Se trata de barras verticales u horizontales con una flecha en cada extremo y un recuadro móvil en su interior, que puede ser desplazado a lo largo de éstas. Estas barras también pueden ser utilizadas para representar magnitudes o permitir que el usuario introduzca un valor dentro de un determinado rango.

Visual Basic dispone de dos tipos de barras de desplazamiento, horizontales y verticales, cuya única diferencia se basa en su orientación física. El nombre de las respectivas clases es *HScrollBar* y *VScrollBar*. La tabla 1 muestra las propiedades, métodos y eventos de estos controles.

Las barras de desplazamiento representan una magnitud dentro de un rango de valores. El valor actual de la magnitud se representa en la propiedad *Value*. Los valores máximo y mínimo de dicho rango se indican en la propiedades *Max* y *Min*, respectivamente. Los valores admitidos van de 0 a 32767.

Como ya se ha mencionado, existen dos tipos de desplazamientos que el usuario puede realizar. El número de unidades en que se incrementa o decrementa la magnitud representada al realizar desplazamientos pequeños o grandes se indica en las propiedades *SmallChange* y *LargeChange*, respectivamente.

Cada vez que la magnitud resulta alterada, ya sea por acción de usuario o a través del código, se produce el evento *Change*, permitiendo así realizar los ajustes oportunos. Sin embargo, cuando la alteración se realiza mediante el arrastre del recuadro móvil, el evento *Change* no se produce hasta que se libera el botón del ratón. En este

caso, puede utilizarse el evento *Scroll*, ya que éste se produce cada vez que cambia *Value*, se haya o no liberado el botón del ratón. El cuadro 1 muestra los pasos para crear una pequeña aplicación ilustrativa, que permite alterar el color de fondo de una etiqueta a partir de los valores de los colores RBG primarios. El aspecto final de la aplicación se muestra en la figura 1.

## TEMPORIZADORES

Cuando en una aplicación se desea realizar una determinada tarea a intervalos fijos de tiempo, se utiliza un control denominado Temporizador (*Timer*). Aunque este control se visualiza en forma icónica en tiempo de diseño, en tiempo de ejecución es invisible. El nombre de la clase es *Timer*, y no dispone de ningún método y tan sólo de un evento.

El funcionamiento de este control es muy similar a la alarma de un reloj. Cada vez que termina un intervalo de tiempo, se produce el evento *Timer*, que se utilizará para realizar las acciones oportunas. El intervalo de tiempo se indica en la propiedad *Interval*, en milisegundos,pudiendo variar entre 1 y 65535. El evento sólo se produce cuando la propiedad *Enabled* se encuentra a *True*, permitiendo así conectar y desconectar el temporizador.

El cuadro 2 muestra los pasos para realizar un pequeño ejemplo ilustrativo del manejo de temporizadores. Se trata de un reloj digital cuya hora se actualiza cada segundo. El aspecto final se muestra en la figura 2.

## CUADROS DE LISTA

Uno de los controles más útiles de Visual Basic son los cuadros de lista (*List box*). Se trata de listas que muestran un grupo de elementos, compuestos de un cierto texto, entre los cuales el usuario puede seleccionar uno o varios, haciendo clic sobre ellos. De esta forma, los elementos seleccionados se muestran resaltados. Cuando el número de elementos es elevado, en la parte derecha aparece un barra de desplazamiento que permite acceder a todos ellos. El nombre de la clase es *ListBox* y sus propiedades, métodos y eventos se muestran en la tabla 2.

La propiedad *ListCount* indica el número de elementos que contiene la lista, mientras que la propiedad *ListIndex* indica la posición del elemento actualmente seleccionado (entre 0 y *ListCount-1*). Si no se encuentra seleccionado ninguno, contendrá -1.

La propiedad *Text* contiene el texto del elemento actualmente seleccionado. Sin embargo, es posible acceder al texto de cualquiera de los elementos a través de la propiedad *List*. Esta propiedad está constituida por una matriz, cuyo índice permite acceder a cada uno de los elementos.

Paralelamente a la matriz *List*, la propiedad *Selected* consiste en otra matriz de valores lógicos que indican si cada uno de los elementos se encuentra o no seleccionado. El número total de elementos seleccionados se indica en la propiedad *SelCount*. Así, por ejemplo, para listar todos los elementos seleccionados, puede utilizarse lo siguiente:

```
For Indice = 1 To List1.ListCount - 1
  If List1.Selected(Indice) Then
    Print List1.List(Indice)
  End If
Next Indice
```

Los cuadros de lista pueden mostrar sus elementos en varias columnas, tal y como se muestra en la figura 3. El número de columnas se indica en la propiedad *Columns*. Los elementos se van situando en las columnas de izquierda a derecha, completando cada fila. Si el espacio disponible no permite visualizar todas las columnas, aparece una barra de desplazamiento horizontal. Si, además, el número de filas que se van llenando es elevado y no es posible visualizarlas todas, también aparecerá una barra de desplazamiento vertical.

Es posible mostrar los elementos de la lista en orden alfabético, aunque sin tener en cuenta la diferencia entre mayúsculas y minúsculas. Para ello, se debe poner a *True* la propiedad *Sorted*. Si *Sorted* se encuentra a *False*, los elementos se muestran en el orden en que fueron añadidos. En el caso de listas ordenadas, el acceso al último elemento añadido puede realizarse a través de la propiedad *NewIndex*.

Los cuadros de lista disponen de tres modalidades de selección distintas. El modo de selección se indica en la propiedad *MultiSelect*, y sus posibles valores son los siguientes:

- 0 (*None*) : Sólo es posible seleccionar un elemento a la vez, haciendo clic sobre él. El elemento anteriormente seleccionado quedará automáticamente deseleccionado. Este es el valor por defecto.
- 1 (*Simple*): Los elementos se seleccionan y deseleccionan haciendo clic sobre ellos, sin que por ello se deseleccionen los elementos anteriormente seleccionados.
- 2 (*Extended*): Permite seleccionar elementos individualmente o en rangos. Para seleccionar un rango debe hacerse clic sobre el primer y último elementos de éste, manteniendo pulsada la tecla *[Shift]*. Si al seleccionar uno o varios elementos se mantiene pulsada la tecla *[Control]*, los elementos anteriormente seleccionados se mantienen.

Para añadir y eliminar los elementos de la lista, se utilizan los siguientes métodos del control:

*-Lista.AddItem Elemento[, Indice]*
Añade un nuevo elemento a la lista, cuyo texto se especifica en el argumento *Elemento*. Opcionalmente, es posible situarlo en una determinada posición indicándola en el argumento *Indice*.

*-Lista.RemoveItem Indice*
Elimina de la lista el elemento cuya posición se indica en el argumento *Indice*.

*-Lista.Clear*
Vacía la lista, eliminando todos los elementos que ésta contiene.

**TABLA 1**

| | |
|---|---|
| Grupo | Repertorio |
| Propiedades | Appearance, Container, DragIcon, DragMode, Enabled, Height, HelpContextID, hWnd, Index, LargeChange, Left, Max, Min, MouseIcon, MousePointer, Name, Parent, SmallChange, TabIndex, TabStop, Tag, Top, Value, Visible, WhatsThisHelpID, Width |
| Métodos | Drag, Move, Refresh, SetFocus, ShowWhatsThis, Zorder |
| Eventos | Change, DragDrop, DragOver, GotFocus, KeyDown, KeyPress, KeyUp, LostFocus, Scroll |

Propiedades, Métodos y Eventos de las barras de desplazamiento.

El cuadro 3 muestra los pasos a seguir para crear un ejemplo ilustrativo del funcionamiento de los cuadros de lista. Se trata una lista en la que pueden añadirse elementos a partir de una cuadro de texto. El aspecto final de la aplicación se muestra en la figura 4.

## CUADROS COMBINADOS

Visual Basic dispone de un tipo de controles, denominados Cuadros combinados (*Combo box*), que permiten combinar las características y el funcionamiento de los cuadros de texto con los cuadros de lista. De esta forma, el usuario puede introducir datos tecleándolos en una zona de edición o seleccionándolos de una lista. El nombre de la clase es *ComboBox*, y la mayoría de sus propiedades, métodos y eventos son comunes a los de los cuadros de texto y los cuadros de lista.

Además de las propiedades comunes a cuadros de texto y cuadros de lista, que funcionan de la misma forma, este control dispone de otra propiedad, denominada *Style*, que define uno de los tres estilos posibles de funcionamiento del cuadro combinado, tal y como se muestra en la figura 5. Los valores posibles son los siguientes:

- 0 (*DropDown Combo*) : Cuadro de texto con un botón que permite desplegar la lista.
- 1 (*Simple Combo*) : Cuadro de texto con la lista siempre desplegada.
- 2 (*DropDown List*) : Sólo se permite seleccionar elementos de la lista desplegable.

Además de los métodos y eventos comunes a los cuadros de texto y los cuadros de lista, cuyo funcionamiento es idéntico, los cuadros combinados disponen también del evento *DropDown*, que se produce cada vez que se despliega la lista.

## CUADROS DE LISTA DE UNIDADES

Visual Basic dispone de unos controles del tipo lista desplegable, especializados en el manejo de las unidades de disco. Se trata de los Cuadros de lista de unidades (*Drive list box*), que permiten seleccionar una de las unidades de disco del sistema (disco duro, disquete, CD-ROM, etc). El nombre de la clase es *DriveListBox*, y sus propiedades, métodos y eventos son muy similares a los del resto de controles de tipo lista.

**TABLA 2**

| | |
|---|---|
| Grupo | Repertorio |
| Propiedades | Enabled, Index, Interval, Left, Name, Parent, Tag, Top |
| Eventos | Timer |

**Propiedades y eventos de los temporizadores.**

A diferencia de la listas desplegables, este tipo de controles no se encuentra inicialmente vacío, sino que contiene tantos elementos como unidades de disco válidas se encuentren disponibles en el sistema. Para seleccionar una de ellas bastará con hacer clic en el elemento correspondiente.

El nombre de la unidad actualmente seleccionada se indica en la propiedad *Drive*, en uno de los siguientes formatos :

- Unidades de disquete : "a:", "b:", etc.
- Unidades de disco duro : "c: [*NombreVolumen*]"
- Conexiones en red : "x:*NombreServidor*"

Si se asigna un valor a través del código, sólo se tendrá en cuenta el primer carácter de la cadena, produciéndose un error si no corresponde a ninguna unidad válida.

## CUADROS DE LISTA DE DIRECTORIOS

Para poder seleccionar y navegar por los distintos directorios y rutas de acceso de una unidad de disco, Visual Basic dispone de otros controles especializados, denominados Cuadros de lista de directorios (*Directory list box*), con aspecto y funcionamiento similares a los de los cuadros de lista estándar. El nombre de la clase es *DirListBox* y sus propiedades, métodos

**CUADRO 1**

1 Añadir al formulario cuatro etiquetas y tres barras de desplazamiento horizontal, situándolas de forma similar a como se muestra en la figura 1.

2 Especificar en tiempo de diseño las siguientes propiedades:

| | | |
|---|---|---|
| Form | Name | frmColores |
| Caption | | Colores de fondo |
| Label1 | Name | lblFondo |
| | Caption | |
| | BorderStyle | 1 - Fixed Single |
| Label2 | Name | lblRojo |
| | Caption | ROJO |
| Label3 | Name | lblVerde |
| | Caption | VERDE |
| Label4 | Name | lblAzul |
| | Caption | AZUL |
| Hscroll1 | Name | hsbRojo |
| | Max | 255 |
| | LargeChange | 5 |
| Hscroll2 | Name | hsbVerde |
| | Max | 255 |
| | LargeChange | 5 |
| Hscroll3 | Name | hsbAzul |
| | Max | 255 |
| | LargeChange | 5 |

3 Escribir el siguiente procedimiento general

```
Private Sub CambiaColor()
  'Combinar los valores de las barras
  lblFondo.BackColor = RGB(hbsRojo.Value, _
    hbsVerde.Value, hbsAzul.Value)
End Sub
```

4 Escribir los siguientes procedimientos de evento:

```
Private Sub Form_Load()
  lblDemo.BackColor = RGB(0, 0, 0)          'Iniciar a negro
End Sub

Private Sub hbsRojo_Change()
  CambiaColor
End Sub

Private Sub hbsVerde_Change()
  CambiaColor
End Sub

Private Sub hbsAzul_Change()
  CambiaColor
End Sub
```

5 Salvar el formulario en el archivo *FONDOS.FRM* y el proyecto en *FONDOS.VBP*.

**Ejemplo de utilización de barras de desplazamiento.**

**CUADRO 2**

1 Añadir al formulario una etiqueta, un botón de comando y un temporizador, situándolos de forma similar a como se muestra en la figura 2.

2 Especificar en tiempo de diseño las siguientes propiedades:

```
Form          Name                          frmReloj
Caption       Reloj digital
Label1        Name                          lblReloj
Caption
BorderStyle   1 - Fixed Single
Font  Size    18
Bold          True
Italic  True
BackColor     &H00000000& - Negro
ForeColor     &H000000FF& - Amarillo
Timer1        Name                          tmrreloj
              Interval          1000
              Enabled           True
Command1      Name                                cmdSalir
Caption       &Salir
```

3 Escribir los siguientes procedimientos de evento:

```
Private Sub tmrReloj_Timer()
  lblReloj = Time              'Leer la hora del sistema
  Beep                                'Emitir un tic
End Sub

Private Sub cmdSalir_Click()
  Unload Me                           'Terminar y salir
End Sub
```

4 Salvar el formulario en el archivo *RELOJ.FRM* y el proyecto en *RELOJ.VBP*.

**Ejemplo de utilización de temporizadores.**

y eventos son similares a los de los cuadros de lista.

Para comprender el funcionamiento de este tipo de listas deben aclararse dos conceptos. No debe confundirse el directorio seleccionado con el directorio actual. El primero es simplemente el directorio que se visualiza resaltado, y puede cambiarse con las teclas del cursor o haciendo clic sobre otro elemento. Sin embargo, el directorio actual es aquel que figura abierto, con todos sus subdirectorios cerrados. Para cambiar de directorio actual, es decir, para situarse en otro directorio, debe hacerse un doble clic sobre él. Esta operación también puede realizarse seleccionando otro elemento y confirmándolo con la tecla *[Intro]*.

La propiedad *Path* indica el nombre del directorio actual. Esta propiedad contiene una cadena con la unidad de disco y la ruta de acceso completa a dicho directorio. Su valor por defecto es la ruta actual de la aplicación en el momento en que se crea el control. Esta propiedad también puede ser especificada a través del código. Así, por ejemplo, para situarse en el directorio de Windows, puede utilizarse la siguiente instrucción:

*Dir1.Path = "C:\Windows"*

A diferencia del resto de controles de lista, la propiedad *ListCount* no indica el número total de elementos de la lista, sino el número de subdirectorios que contiene el directorio actual. Igualmente, la propiedad *ListIndex* indica el directorio que se encuentra seleccionado o resaltado y, en principio, puede no coincidir con el directorio actual. Además, los valores de *ListIndex* no varían entre 0 y *ListCount-1*. Los directorios abiertos, es decir, el directorio actual y sus directorios padre, se corresponden con valores negativos de *ListIndex*. Por su parte, los directorios cerrados, es decir, los subdirectorios del actual, se corresponden con valores positivos. Por tanto, el valor para el directorio actual será -1, mientras que para su primer directorio padre será -2, y así sucesivamente. Los valores para los subdirectorios del actual estarán comprendidos entre 0 y *ListCount-1*. Igualmente, las cotas superior e inferior de la matriz *List* vendrán determinadas por el rango de valores actual de *ListIndex*.

Cada vez que se cambia de directorio actual, es decir, al cambiar el valor de la propiedad *Path*, se produce el evento *Change*. Sin embargo, si sólo se cambia de directorio seleccionado, es decir, al cambiar el valor de *ListIndex* y no el de *Path*, sólo se produce el evento *Click*.

## CUADROS DE LISTA DE ARCHIVOS

Para visualizar nombres de archivos, de forma que el usuario pueda seleccionarlos, Visual Basic dispone de un control, denominado Cuadro de lista de archivos (*File list box*). Se trata un control muy similar a los cuadros de lista estándar, en el que se visualizan los nombres de una serie de archivos, pertenecientes todos ellos a un determinado directorio. El usuario puede seleccionar uno o varios entre ellos, visualizándose entonces resaltados. El nombre de la clase es *FileListBox*, y la mayoría de sus propiedades, métodos y eventos son comunes a los cuadros de lista estándar.

La propiedad *Path*, sólo disponible en tiempo de ejecución, indica la ruta de acceso completa al directorio de los archivos. Así, por ejemplo, para visualizar los archivos del directorio de Windows, puede utilizarse la siguiente instrucción:

*File1.Path = "C:\Windows"*

La propiedad *Pattern* indica un patrón de nombre que deben cumplir los archivos que se visualizarán en la lista. Para ello es posible utilizar los caracteres comodín del sistema operativo (? y *). Su valor por defecto es "*.*", indicando que se visualizarán todos los archivos del directorio. Así, por ejemplo, para visualizar sólo los archivos de texto puede utilizarse la siguiente instrucción:

*File1.Pattern = "*.TXT"*

La propiedad *FileName*, disponible sólo en tiempo de ejecución, contiene el nombre del archivo de la lista que se encuentra actualmente seleccionado. Su valor inicial en el momento de la creación de la lista es "", indicando que no se encuentra seleccionado ningún archivo. Cuando se lee su valor en el código, devuelve el nombre del archivo seleccionado, sin incluir la ruta de acceso. Sin embargo, a la hora de asignar su valor en el código, es posible incluir la unidad, la ruta de acceso y un patrón de nombre. De esta forma, se actualizará tanto el valor de *Path* como el de *Pattern*. Si simplemente se especifica un nombre de archivo sencillo, sin alterar la ruta de

**CUADRO 3**

1 Añadir al formulario un cuadro de lista, un cuadro de texto y cuatro botones de comando, situándolos de forma similar a como se muestra en la figura 4.

2 Especificar en tiempo de diseño las siguientes propiedades:

```
Form        Name                        frmLista
Caption     Demostración de lista
List1       Name                        lstDemo
            Multiselect 2 - Extended
Text1       Name                        txtDemo
            Text
Command1                Name                        cmdAñadir
            caption                 &Añadir
Command2                Name                        cmdEliminar
            caption                 &Elminar
Command3                Name                        cmdBorrar
            caption                 &Borrar todo
Command4                Name                        cmdSalir
            caption                 &Salir
```

3 Escribir los siguientes procedimientos de evento:

```
Private Sub cmdAñadir_Click()
  lstDemo.AddItem txtDemo.Text          'Añadir texto
  txtDemo.Text = ""                                 'Borrar contenido
End Sub

Private Sub cmdEliminar_Click()
  Dim Ind As Integer
  'Salir si lista vacía
  If lstDemo.ListCount = 0 Then Exit Sub
  'Si elemento seleccionado
  For Ind = 0 To lstDemo.ListCount - 1
    If lstDemo.Selected(Ind) Then
      lstDemo.RemoveItem Ind            'Eliminarlo
    End If
  Next Ind
End Sub

Private Sub cmdBorrar_Click()
  lstDemo.Clear                                     'Borrar todos
End Sub
Private Sub cmdSalir_Click()
  Unload Me
End Sub
```

4 Salvar el formulario en el archivo *LISTA.FRM* y el proyecto en *LISTA.VBP*.

**Ejemplo de utilización de cuadros de lista.**

**CUADRO 4**

1 Añadir al formulario una etiqueta, un cuadro de lista de unidades, uno de directorios, y uno de archivos, situándolos de forma similar a como se muestra en la figura 6.

2 Especificar en tiempo de diseño las siguientes propiedades:

```
Form    Name                    frmTamañoCaption        Visor de tamaños
Label1  Name                    lblTamaño
        Caption
        Alignment       2 - Center
        BorderStyle     1 - Fixed Single
        Font            Bold                    True
Drive1  Name                    drvArchivo
Dir1    Name                    dirArchivo
File1   Name                    filArchivo
```

3 Escribir los siguientes procedimientos de evento:

```
Private Sub dirArchivo_Change()
  filArchivo.Path = dirArchivo.Path
End Sub
Private Sub drvArchivo_Change()
  dirArchivo.Path = drvArchivo.Drive
End Sub
```

4 Salvar el formulario en el archivo *TAMAÑOS.FRM* y el proyecto en *TAMAÑOS.VBP*.

**Ejemplo de utilización conjunta de las listas de unidades, directorios y archivos.**

acceso, el archivo quedará seleccionado.

Las propiedades *Archive*, *Hidden*, *Normal*, *ReadOnly* y *System* indican mediante valores lógicos si la lista de archivos visualizará archivos con los atributos del sistema operativo "Modificado", "Oculto", "Normal", "Sólo Lectura" y "Sistema", respectivamente. De esta forma, es posible visualizar sólo un determinado tipo de archivos, en función de los valores de sus atributos. El valor por defecto para *Archive*, *Normal* y *ReadOnly* es *True*, mientras que para *Hidden* y *System* es *False*. Así, por ejemplo, para visualizar los archivos de sólo lectura puede utilizarse la siguiente instrucción:

*File1.ReadOnly = False*

Cuando en el código se cambia la ruta de acceso a los archivos de la lista, ya sea especificando un valor nuevo en *Path* o en *FileName*, se produce el evento *PathChange*. Igualmente, cuando se altera el patrón del nombre de los archivos, ya sea especificando un valor nuevo en *Pattern* o en *FileName*, se produce el evento *PatternChange*.

## UTILIZACIÓN CONJUNTA

Utilizando de forma conjunta un cuadro de lista de unidades, uno de directorios y uno de archivos, es posible hacer que el usuario pueda moverse y seleccionar archivos por todas las unidades y directorios del sistema. Para ello, será necesario sincronizar su funcionamiento, de forma que los cambios realizados en cada uno de los controles sean automáticamente reflejados en el resto.

Cuando el usuario seleccione una unidad de disco en el cuadro de lista de unidades, el cuadro de lista de directorios debe reflejar la estructura de directorios de dicha unidad. Para que esto ocurra, puede utilizarse el procedimiento de evento *Change* de la lista de unidades de la siguiente forma:

```
Private Sub Drive1_Change()
  Dir1.Path = Drive1.Drive
End Sub
```

Igualmente, cuando el usuario cambie de directorio en la lista de directorios, la de archivos debe contener los archivos correspondientes de ese directorio. Para que esto ocurra, puede utilizarse el procedimiento de evento *Change* de la lista de directorios de la siguiente forma:

```
Private Sub Dir1_Change()
  File1.Path = Dir1.Path
End Sub
```

El cuadro 4 muestra un ejemplo ilustrativo de la utilización conjunta de estos tres tipos de lista.

# CONTENIDO DEL CD-ROM

## DISTRIBUCIÓN TCL/TK PARA WINDOWS 3.1, WINDOWS 95 WINDOWS NT, DOS, OS/2 Y MAC

El CD-ROM de regalo de este mes está dedicado a Tcl/Tk. En él se incluyen las distribuciones Tcl/Tk para Unix, Windows 3.1, 95 y NT, DOS, Macintosh y OS/2, las bibliotecas y extensiones más importantes que se han desarrollado, así como ejemplos y la documentación necesaria para iniciar el desarrollo inmediato de aplicaciones en este nuevo lenguaje.

El TCL, de las iniciales de *Tool Command Language,* es básicamente otra shell más del sistema operativo Unix, aunque cuenta con una semántica propia. La novedad de esta shell es que está asociada a un *toolkit*, el Tk, para el desarrollo de aplicaciones visuales en el ambiente de las X-Windows, por lo que proporciona una potencialidad enorme para el desarrollo de sistemas gráficos.

El lenguaje Tcl nació a principios de la década pasada pero no fue hasta 1988 que John Ousterhout, profesor de la Universidad de California en Berkeley, le adecuó un *toolkit* con bibliotecas gráficas buscando una herramienta para prototipado rápido de interfaces de usuario.

Desde su uso generalizado a principios de 1993, siempre se le ha atribuido cierta similitud con el Visual Basic de Microsoft, ésto se debe en parte a que al igual que éste tiene una gran facilidad para la creación de aplicaciones completas en tiempos de desarrollo cortos y que a ambos se les considera como lenguajes interpretados.

Como se ha mencionado anteriormente, TCL es básicamente una shell de Unix, pero a causa del gran éxito que ha tenido en ese ambiente su versión más reciente, la 7.5 de Tcl y 4.1 de Tk, se ha portado "oficialmente" a otras plataformas. Es por esto que ahora es posible migrar una aplicación Tcl/Tk que originalmente fue diseñada para trabajar en Linux a Windows 95 o viceversa. Esta nueva versión incluye por primera vez en el mismo lenguaje comandos de comunicaciones a través de la creación de sockets.

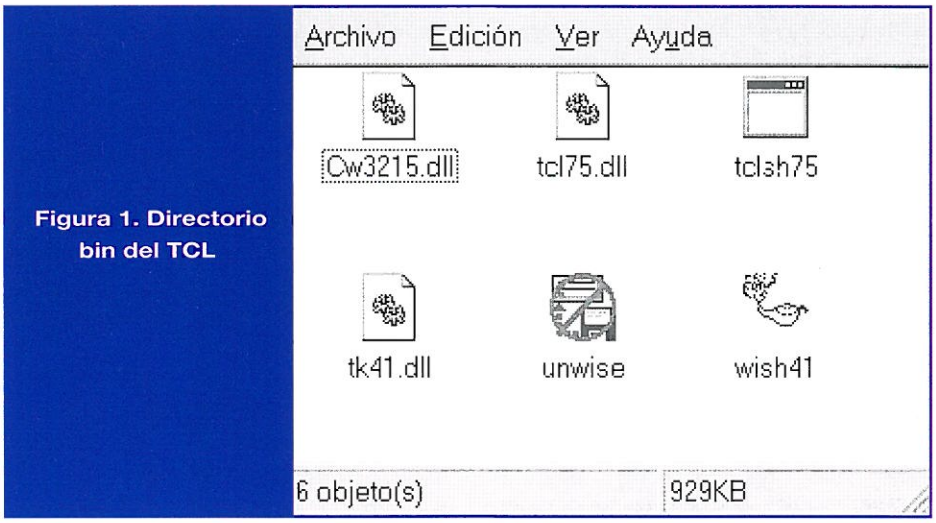



**Figura 1. Directorio bin del TCL**

En la actualidad, el Tcl también se ha hecho un lugar entre los lenguajes de programación en Internet, principalmente en la creación de CGIs (*Common Gateway Interface*), que son tan populares en las páginas interactivas del WWW. El CD-ROM contiene algunos ejemplos de programación de CGIs. Además, la versión 7.5 incluye algunas características de seguridad para Internet, por lo que en algunos ambientes a esta versión se le conoce como *safe-tcl.*

Se recomienda antes de proceder a la instalación visualizar el fichero Indice.htm que se encuentra en el directorio Tcltk para lo cual es necesario el

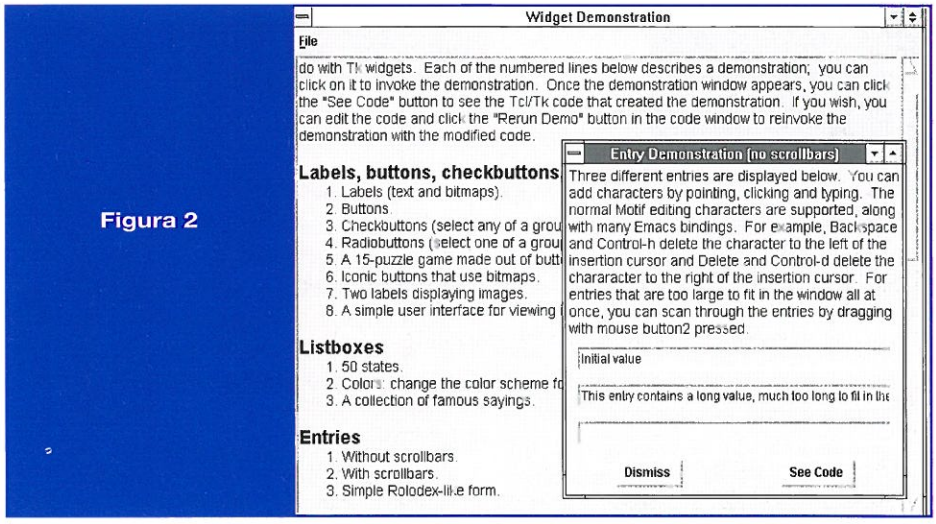



**Figura 2**

empleo de un Browser como Netscape o Internet Explorer de Windows 95. De cualquier manera en el directorio Browsers se encuentra entre otros Netscape.

## INSTALACIÓN EN WINDOWS 95 y NT

La versión de Tcl/Tk para Windows 95 se puede instalar directamente desde el mismo CD-ROM, para ello es necesario entrar al directorio Tcltk\Windows del CD-ROM y hacer doble click sobre el icono del fichero *"win41.exe"*, lo que iniciará al programa de instalación, bastará con seguir las instrucciones que aparecerán en la pantalla para culminar la instalación.

## INSTALACIÓN EN WINDOWS 3.1

Para poder instalar el Tcl/Tk en Windows 3.1 será necesario que esté instalada la biblioteca para el manejo de programas de 32 bits, si no está instalada, será necesario copiar el fichero *w32s125.exe*, que se encuentra en Tcltk\Windows a un directorio temporal y descomprimirlo para ello bastará con ejecutar *"w32s125.exe"*, una vez que haya finalizado la descompresión se podrá instalar la biblioteca ejecutando *"setup.exe"*. Posteriormente sólo restará repetir los pasos de la instalación para Windows 95.

Para ejecutar el tcl, ver Figura 1, será necesario ejecutar el comando *wish41* que estará dentro del directorio "bin" de Tcl (el directorio Bin se crea en el proceso de descompresión), y todo estará preparado para empezar a trabajar.

## INSTALACIÓN EN LINUX

Para instalar la distribución que funciona en Linux, es recomendable utilizar la versión 7.4 del Tcl y 4.0 del Tk, con sus respectivos *patches*, ya que en ésta se pueden utilizar la mayoría de las extensiones existentes para Tcl/Tk. Aunque bien, los pasos a seguir son exactamente los mismos para cualesquiera de las versiones que hay en la distribución.

Para la versión 7.4 del Tcl y 4.0 del Tk los ficheros necesarios para realizar la instalación son: *tcl74.tgz* y *tk40.tgz*, y sus respectivos patches que son los ficheros *tcl74pat.zip* y el *tk40pat.zip*, localizados en el directorio \Tcltk\Unix.

Primero será necesario crear un directorio en el \usr\local que bien puede llamarse *tcl-tk*, y copiar en él los ficheros arriba mencionados. Posteriormente, será necesario expandir los ficheros, *tcl74.tgz* y *tk40.tgz*, para esto sólo será necesario realizar las siguientes dos operaciones en la línea de comandos:

*dstar:/usr/local/tcl-tk>gzip -d < tcl74.tgz | tar -xvf -*
*dstar:/usr/local/tcl-tk>gzip -d < tk40.tgz | tar -xvf -*

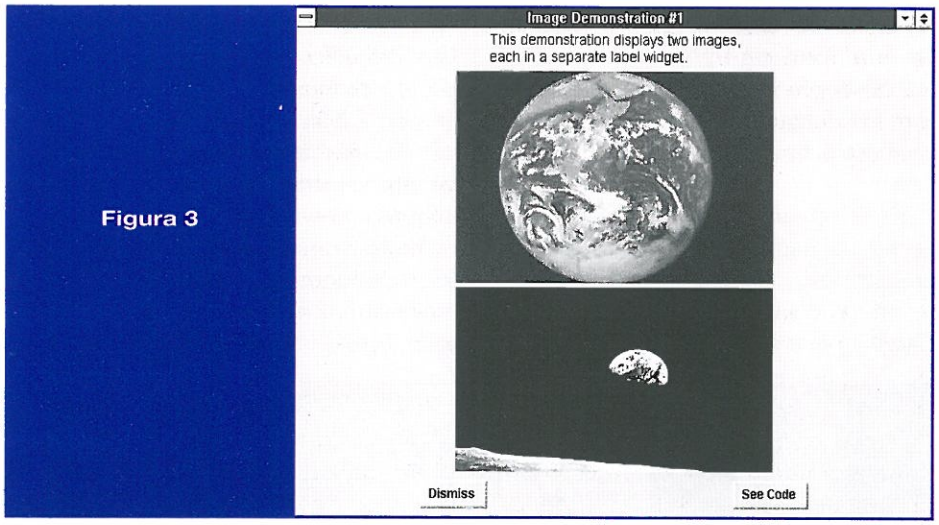



Figura 3

con esto se crearán los directorios *tcl7.4* y *tk4.0* respectivamente. El siguiente paso será aplicar los patches, para ello será necesario descomprimir los ficheros *tcl74pat.zip* y *tk40pat.zip* en sus respectivos directorios (tcl7.4 y tk4.0), una vez que ya que hayan descomprimido estos dos ficheros será necesario aplicar los patches, para lo cual sólo habrá que realizar el comando patch a cada uno de ellos:

### BIBLIOGRAFÍA


- Brent B. Welch, Practical Programming in Tcl and Tk, Prentice Hall, 1995
- John Ousterhout, Tcl and the Tk Toolkit, Addison-Wesley, 1994.
- News
news:comp.lang.tcl
- Tcl/Tk Resources
http://web.cs.ualberta.ca/~wade/Auto/Tcl.html
- Tcl WWW Info
http://www.sco.com/Technology/tcl/Tcl.html


*dstar:/usr/local/tcl-tk/tcl7.4> patch -d < tcl7.4p1.patch*
*dstar:/usr/local/tcl-tk/tcl7.4> patch -d < tcl7.4p2.patch*
*dstar:/usr/local/tcl-tk/tcl7.4> patch -d < tcl7.4p3.patch*

y

*dstar:/usr/local/tcl-tk/tk4.0> patch -d < tk4.0p1.patch*
*dstar:/usr/local/tcl-tk/tk4.0> patch -d < tk4.0p2.patch*
*dstar:/usr/local/tcl-tk/tk4.0> patch -d < tk4.0p3.patch*

una vez realizada esta operación será necesario configurar la distribución, para esto se deberá ejecutar dentro de *tcl7.4*:

*dstar:/usr/local/tcl-tk/tcl7.4>./configure*

y una vez que haya terminado la configuración habrá que ejecutar el comando *"make"*. Una vez que haya terminado de compilar el Linux, cambie al directorio *tk4.0* y repita la operación ejecutando primero *./configure* y luego *"make"*.

*dstar:/usr/local/tcl-tk/tk4.0>./configure*
*dstar:/usr/local/tcl-tk/tk4.0>make*

Una vez compilado el Tcl y el Tk, se podrán ejecutar comandos de Tcl invocando el comando tclsh7.4 y el Tcl/Tk completo invocando el comando wish4.0, para esto aparecerá una ventana pequeña y la línea de comandos estará lista para recibir cualquier comando de tcl.

## FUENTES DE LOS ARTÍCULOS

Como es habitual en el directorio Disk24 se encuentran los fuentes y utilidades de los artículos de este mes.

# CORREO DEL LECTOR

**En esta sección, los lectores de SÓLO PROGRAMADORES tienen la oportunidad de hallar respuesta a los problemas que puedan tener en cualquier tema relacionado con la programación.**

## MÁS SECCIONES EN LA REVISTA

*Hola amigos de Sólo Programadores, hace muy poco que conozco vuestra revista, pero me parece bastante interesante, lo malo es que yo llevo relativamente poco en esto de la programación, he empezado este año a estudiar Pascal, mi pregunta es la siguiente, ¿por qué no dedicáis más secciones a los principiantes como yo para que podamos llegar a tener el nivel suficiente para aprovechar todos los cursos que dais en vuestra revista?*

*Javier Varela*
*( Espinosa/Guadalajara)*

Lo primero gracias por tu carta y tus comentarios, la verdad es que es muy difícil hacer una revista de programación a gusto de todos, ya que hay gente que quiere muchos artículos de bajo nivel, mientras otros quieren más de comunicaciones (redes, Internet) o programación visual. Nosotros intentamos complacer a todos y sobre todo mantenernos en vanguardia, es decir estar informados de todas las novedades y avances en cualquier área de programación para inmediatamente transmitirlo a nuestros lectores. En cuanto a la gente principiante, siempre tenemos una sección, ahora Curso de Programación, en el cual se ha ido avanzando desde una introducción a la programación hasta el estudio de distintos algoritmos de programación, eso sí siempre en pseudocódigo menos los últimos capítulos en los cuales se han codificado ciertos algoritmos en Pascal.

De cualquier manera amigo Javier no descartamos en próximos números añadir alguna sección más orientada exclusivamente a principiantes.

## DESPLAZAMIENTO EN OPERACIONES MATEMÁTICAS

*Hace varios números en un artículo de vuestra revista sobre cómo optimizar código en ensamblador se afirmaba que es mucho más eficiente a la hora de efectuar ciertas operaciones matemáticas como una multiplicación, utilizar un desplazamiento en vez del operador matemático de multiplicación. Yo no entiendo mucho de programación a tan bajo nivel pero me gustaría que me explicarais un poco este tema.*

*Sergio Rodríguez*
*(Coslada / Madrid)*

Las instrucciones de multiplicar MUL e IMUL la primera para números con signo y la segunda para números sin signo te podemos decir que la más rápida, la primera, tarda a partir de 10 ciclos en un Pentium, en un 486 más de 27 ciclos, en cambio las operación de rotar y sumar registros tardan 1 ciclo cada una. Por lo tanto, para el caso de operaciones por potencia de dos parece evidente la ganancia de tiempo ya que multiplicar por potencias de dos equivale a un desplazamiento por dicho número hacia la izquierda. Veamos un ejemplo

```
7 * 2 ^ 5 = 7 rot 5
MUL 32 => SHR EAX, 5
```

En el caso de no ser potencia de 2 se descomponen en sumas de potencias de 2 ya que podríamos hacer hasta 5 desplazamientos y 5 sumas, y tardaríamos lo mismo que en la multiplicación

```
6 * 27 = 6 * (16+8+2+1) = 6 * (2^4+2^3+2+1) = 6 * 2 ^ 4 + 6 * 2^ 3 + 6*2 + 6
```

3 sumas y tres desplazamientos que hacen un total de 6 ciclos frente a los teóricos diez de la multiplicación, si además tenemos en cuenta que el Pentium ejecuta determinadas instrucciones de dos en dos, el número de ciclos disminuiría aún más por lo que la velocidad sería mayor en contra de las afirmaciones que se han oído por Internet sobre el hecho de que en un Pentium es mejor multiplicar que rotar ( el Pentium Pro funciona de otra manera y de ahí a podido venir el problema).

## NÚMEROS ATRASADOS

*Os escribo porque, lamentablemente, tan sólo hace un año que conozco y compro vuestra revista. Me interesan mucho los temas que en ella tratáis, por ello, me gustaría poder tener todos los números que habéis editado, ya que no me ha sido posible conseguir los números 1 al 12 mediante los distribuidores. Por favor, ¿cómo puedo conseguir dichos números?*

*Ana Isabel Santana Toledo*
*(Málaga)*

Lamentamos que no hayas podido obtener los números atrasados que mencionas a través de tu distribuidor habitual. Sin embargo, siempre puedes pedirlos a la propia editorial. Puedes enviar una carta con tu pedido a la siguiente dirección:

Tower Communications S.R.L.
Dpto. De suscripciones
C/ Aragoneses 7
Pol. Ind. Alcobendas 28.100 (Madrid)

También puedes llamar al teléfono (91) 661 42 11 o enviar un fax al número (91) 661 43 86.

## ¿CUÁL ES EL MEJOR COMPILADOR?

*Estimados amigos de Sólo Programadores, os escribo porque tengo una duda rondándome la cabeza. Estoy siguiendo con mucha atención vuestra serie de artículos sobre "Cómo programar una demo" y estoy pensando en utilizar los conocimientos adquiridos en la creación de un juego. Sé que hay muchos compiladores en el mercado, pero no sé cual es el mejor para escribir un juego basado en MS-DOS, ya que apenas tengo conocimientos de programación bajo Windows. ¿Qué compilador me recomendáis?*

*Javier Losada*
*(Sevilla)*

Amigo Javier, has acudido al lugar idóneo para resolver tu problema. En efecto, existen varios compiladores y entornos de desarrollo de gran calidad en el mercado, por lo que es perfectamente comprensible que te hayan surgido dudas al respecto. La mayoría de los compiladores actuales son completos entornos de desarrollo capaces de crear aplicaciones de todo tipo para distintas plataformas. En principio cualquiera de ellos te podría servir, pero si lo único que quieres es crear juegos la elección más adecuada es el compilador Watcom C++. Este entorno incorpora un extensor MS-DOS de 32 bits ampliamente utilizado por los programadores de juegos para MS-DOS ¿te suena la cabecera "DOS/4GW" que aparece al cargar la mayoría de ellos?

Además, Watcom C++ incluye un ensamblador para la creación de rutinas a muy bajo nivel. Si quieres utilizar buenos efectos para mejorar la estética de tu juego, no tendrás más remedio que utilizarlo.

## LIBROS DE PROGRAMACIÓN EN C++

*Hola, soy un asiduo lector de su revista que desea aprender todos los secretos de la programación en C++. He visto muchos libros de C++ en las librerías, pero no sé cual de ellos debo comprar sin llevarme luego una gran decepción. Mi pregunta es la siguiente: ¿Cuál es el mejor libro a la venta para convertirse en un profesional de la programación en C++?*

*Luis del Río*
*(Albacete)*

La respuesta a su pregunta depende mucho de su nivel de conocimientos en materia de programación. El lenguaje C++ es una mejora considerable de C, con orientación a objetos y otras características avanzadas. Por ello, si usted maneja con soltura el lenguaje C le será muy fácil aprender C++. En este caso le podemos recomendar la obra "C++. Guía de autoenseñanza" de Herbert Schildt publicada por la editorial McGraw-Hill, y que fue comentada en el número 9 de Sólo Programadores.

Si por el contrario desconoce por completo el lenguaje C, tiene dos opciones. La primera es aprender C en primer lugar con un libro como "¡Aprenda C ya!" de Augie Hanson (editorial Anaya) y después utilizar el libro de Schildt. La segunda opción es aprender C++ directamente, para la cual le puede servir perfectamente el libro "Como programar en C++" de H. M. Deitel.

De todos, las lecturas obligadas para todo buen programador de C y C++ son "El lenguaje de programación C" de Kernighan y Ritchie, y "El lenguaje de programación C++" de Bjarne Stroustrup. Estos son libros muy buenos, pero no es recomendable su lectura sin haberse iniciado previamente en la programación en ambos lenguajes.

## EL GAME SDK

*Hola, llevo poco tiempo programando bajo Windows 95 y creo que no es tan bonito como lo pintan, ya que el API no me permite todo el control que a mí me gustaría, porque vengo de programar en DOS y allí la libertad sobre el sistema es total. He oído hablar del Game SDK de Microsoft y me gustaría conocer qué ventajas me ofrece este producto.*

*David Nadal*
*(Madrid)*

El GSDK de Microsoft, también llamado Direct X, ofrece al programador la posibilidad de saltarse tanto el API como el GUI de Windows y permite el uso más o menos directo de los recursos del sistema. El paquete está dividido en cuatro partes: Direct Draw, para manejo de modos de video, paginación, paletas, etc; Direct Sound, mezcla y reproducción de sonidos; Direct Play, que permite la programación de sistemas multiusuario (módem, IPX, TCP/IP) y Direct Input, que permite acceso directo a dispositivos de entrada(Joystick). Existen, además, actualizaciones a este paquete de próxima aparición, como son Direct Video, que facilitará el uso de AVIs y otros formatos de video en las aplicaciones, y Direct 3D, que proporcionará un soporte para la programación 3D.

# ¡¡NO ESPERES MAS!!

## PARA ACCEDER A LAS

# AUTOPISTAS

## DE LA ¡CONECTATE!

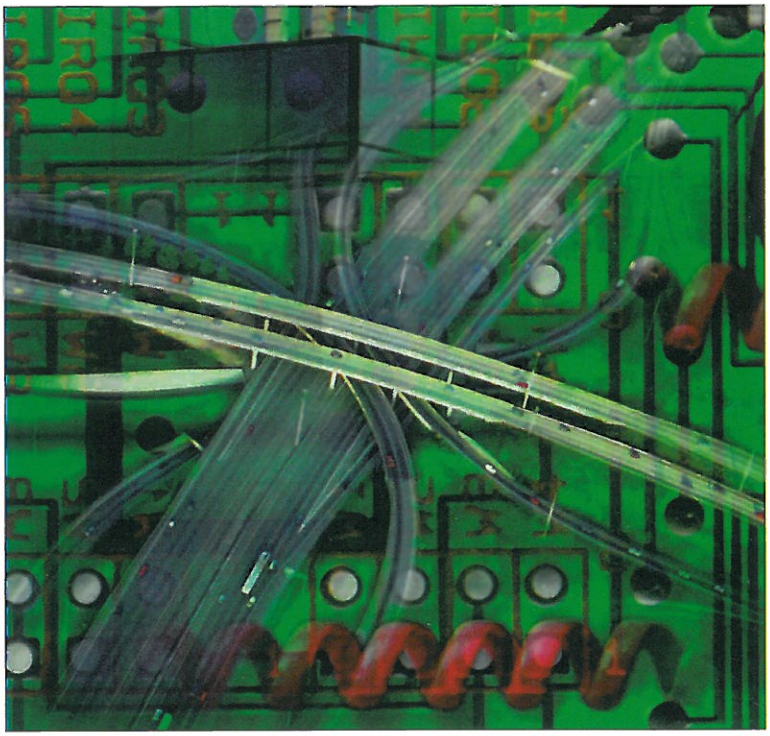
# INFORMACION

PORQUE LA INFORMACION ES PODER Y PORQUE QUIERES SABER MAS, NO PUEDES ESTAR DESCONECTADO DE MAS DE 40.000.000 DE PERSONAS DE TODO EL MUNDO QUE PIENSAN COMO TU. Y QUE SABEN, QUE LA INFORMATICA NOS AYUDA A ELIMINAR BARRERAS , Y A ENTENDERNOS MAS Y MEJOR.

**AHORA PUEDES CONECTARTE A INTERNET A UN PRECIO EXCEPCIONAL**

**¡¡¡ 1.995 PTAS. AL MES !!!**

- SIN CUOTA DE CONEXION.
- SIN LIMITE DE TIEMPO.
- SIN RETRASOS NI ATASCOS.
- INCLUYENDO SOFTWARE INFOVIA-INTERNET



Envíanos este cupón de solicitud de conexión y recibirás el software y una domiciliación bancaria. Remitir a Eurotex c/ Augusto Figueroa 32-4º E Madrid-28004 - Tfno. (91) 523 18 93 (de 19 a 21h.)

---

Nombre. . . . . .

Dirección . . . . . .

Población . . . . . Provincia . . . . . .

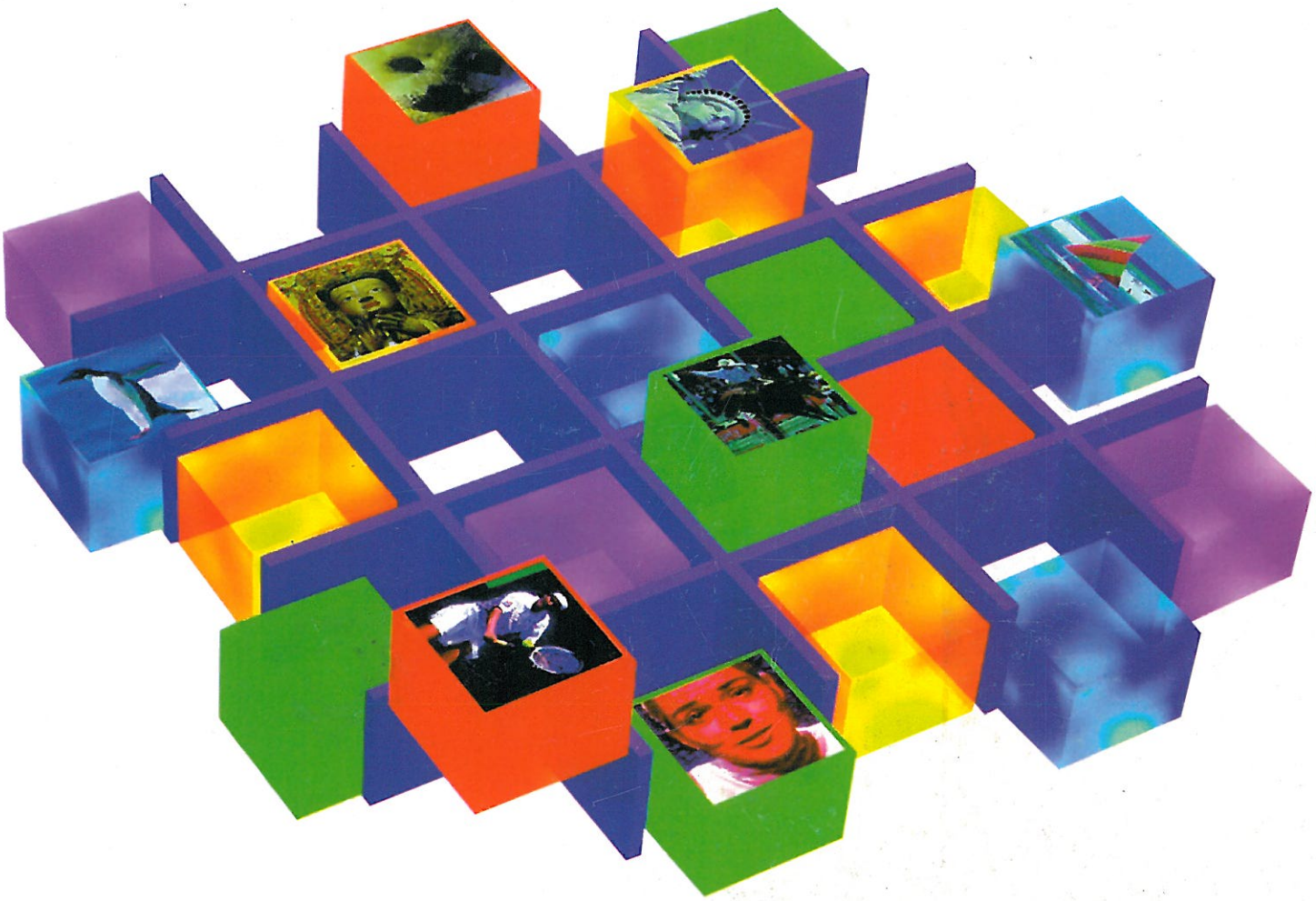
CP. . . . . Teléfono . . . . . .

# PROGRAMACION WINDOWS

## CURSO PRACTICO DE

# Visual Basic 4.0

**PROGRAMACION EN 32BITS. LA PROGRAMACION DEL FUTURO.**



Con la colaboración de Microsoft®

PARA MÁS INFORMACIÓN LLAMAR AL TÉLF.: (91) 661 42 11

ABETO EDITORIAL